(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 June 2002 (27.06.2002)

PCT

(10) International Publication Number
WO 02/50691 A1

BEST AVAILABLE COPY

(51) International Patent Classification[7]: G06F 15/00, 17/00, 17/21, 17/24, 17/60

(21) International Application Number: PCT/US01/50279

(22) International Filing Date:
19 December 2001 (19.12.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/256,823 19 December 2000 (19.12.2000) US
60/256,794 19 December 2000 (19.12.2000) US

(71) Applicant: MEDIAGATE, INC. [US/US]; 1245 South Winchester Boulevard, Suite 208, San Jose, CA 95128 (US).

(72) Inventors: HULL, Chris. BAUMANN, Cliff; 20601 Henwood Road, San Jose, CA 95120 (US). FRANCIS, Brett; 232 Jeter Street, Redwood City, CA 94062 (US). GAURAV, Kohli.

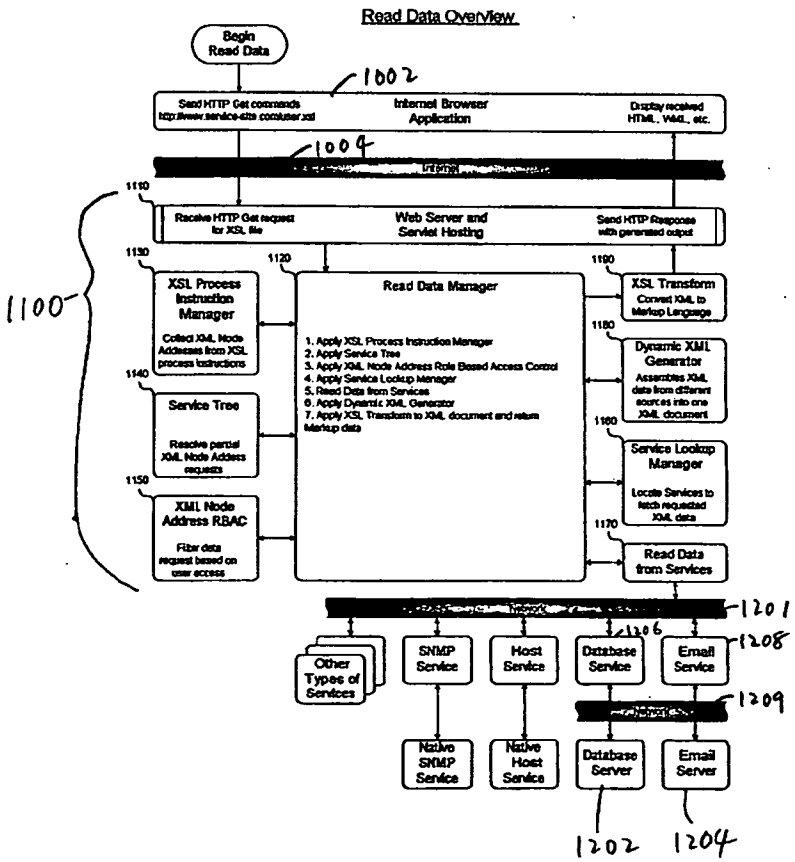
(74) Agent: HSUE, James, S.; SKJERVEN MORRILL LLP, Three Embarcadero Center, 28th Floor, San Francisco, CA 94111 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

[Continued on next page]

(54) Title: SOFTWARE ARCHITECTURE FOR INTERACTION WITH DYNAMIC DATA SOURCES AND ROLE BASED ACCESS CONTROL



(57) Abstract: The invention uses dynamically generated and collated Extensible Markup Language (XML) data to enable access to disparate data sources in a uniform way. The invention uses dynamic data source resolution to generate data requests on behalf of a specific application interface (1002). Role based access control is employed to filter the request based on pre-defined permissions employing XML node addresses. Then the invention relies upon a dynamic XML generator to receive the data response from the Internet (1004) and collate XML data from the multiplicity of available data sources taken from services like a Database service (1206) or Email Service (1208). The dynamic XML generator uses the collated data to dynamically create a uniform formatted representation of the received data. The invention understands a data source as any single software entity the dynamic XML generator can interact with to receive application specific data in an XML format.

WO 02/50691 A1

patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SOFTWARE ARCHITECTURE FOR INTERACTION WITH DYNAMIC DATA SOURCES AND ROLE BASED ACCESS CONTROL

BACKGROUND OF THE INVENTION

[0001] This invention relates to an apparatus and method for dynamically generating and collating data to enable access to disparate data sources in a uniform manner.

[0002] In a client-server software architecture where application components are defined through the use of Java Servlets, CGI scripts, and generally any software interface module, components must be tailored to specifically access databases and other server-side data sources as well as provide interfaces, which are specific to those applications. This common architecture, while somewhat flexible, requires a great deal of engineering effort in order to change the behavior of any interface and to maintain interface compatibility whenever back-end data sources change. This software architecture is commonly deployed by many websites today. Extensible Markup Language ("XML") provides a uniform mechanism for representing data. Extensible Style Language (XSL) transformations provide a flexible and easy to use mechanism for transforming XML data into markup-based user interface data (e.g. HTML and WML) or any XML describable data. What is missing is a mechanism for requesting data from loosely associated disparate data sources that collates data from any data source into a single XML representation. This single representation is then usable for processing and display while maintaining fine-grained resolution and security. Information concerning XML can be found in REC-xml-19980210 "Extensible Markup Language (XML) 1.0"

SUMMARY OF THE INVENTION

[0003] This invention is based on the observation that the above-described difficulties may be alleviated by providing a common XML-related addressing mechanism through which disparate data sources may provide data to a software entity requesting data or service. By providing a common mechanism for users to request data from loosely associated disparate data sources, there is no need for the software entity to have access to the particular interfaces designed for each data source, or even knowledge of such interfaces. In one embodiment, the common XML-related addressing mechanism comprises XML node addresses and a common vehicle for obtaining from these addresses the appropriate data sources for providing the service or data requested.

[0004] In the same vein, a common XML-related addressing mechanism may be provided through which disparate data sources may be identified from a request from a software entity as the ones to which information is to be written, and the information is then written to the data sources so identified. In one embodiment, the common XML-related addressing mechanism comprises XML node addresses which may be read from the request for identifying the data sources.

A mechanism is provided as an interface in a computer system for transfer of information to and/or from a group of data sources that supply data to serve request(s) by a software entity. The interface comprises a common addressing mechanism for the group of data sources. Preferably the mechanism comprises XML hierarchical data structures and a common method for accessing the structures.

[0005] To ensure that the user requesting data or service has the appropriate permission for such data or service, the role of the user is checked before the common mechanism returns a set of data sources that provide the data or service. Checking the permission of the user at the front end reduces traffic and enhances efficiency of the system. In one embodiment, a request by the user for data or service is phrased in the form of XML node addresses. The role based access control (RBAC) system resolves these node addresses and compares the request to the role profile of the user and filters the request to return the permitted data and service to the user. Preferably, the role based access control system returns a set of data sources that provide data and

service that fit the role profile of the user.

[0006] To enable user requests to be made through simplified XML node address formats, a truncated or partial XML node address employing one or more shorthand characters may be employed in the requests. These partial XML node addresses are resolved with the aid of service trees of full XML node addresses for compiling a full list of XML node addresses. The node or nodes in the service tree corresponding to the shorthand character are found, and all progeny nodes to such node identified and collected to form a list of XML node addresses.

[0007] A XML mapping mechanism is provided comprising a service tree. The service tree comprises one or more service type nodes, and at least one additional node that is a child of one of said one or more service type nodes. Preferably the service tree comprises one or more leaf nodes that are children of the at least one additional node.

[0008] A mapping mechanism may be used for finding data sources that provide services requested by a software entity. First a XML node address submitted by the software entity is parsed. A match in the service tree to the XML node address is found and at least one data source that provides services requested by the software entity is located. The mapping mechanism used is preferably the one outlined above.

[0009] The data sources provide to the system the XML node addresses by which it can be accessed.

[0010] Preferably, the system resolves the XML node addresses resulting from the user request to obtain a list of data sources that can provide the service or data requested by the user. Preferably, the resolution is performed by a service lookup manager using a service tree.

[0011] When multiple sets of data are returned directly to the user from two or more data sources in response to a user request, the multiple sets of data may be difficult for the user to process and use. Thus, another aspect of the invention is based on the recognition that where the data returned by multiple data sources are in the form of XML representations, these representations may be collated before they are returned to the user so that the data is much easier to process and use by the user.

This is performed by a XML generator. In the preferred embodiment, the XML generator matches corresponding nodes in the XML representations from different data sources. Where the XML representations returned by the data sources comprise XML node addresses, preferably these representations are collated into a single XML representation by building a XML tree, where the building process resolves the XML node addresses.

[0012] The above described methods, data structures and mechanisms may be implemented as software. The software may be stored in any storage medium and transported. When the software in the medium is loaded into a computing device, the device can then carry out the various functions described above and herein. Such software may also be carried by a signal that is transmitted through wires, wireless or optical channels. When the signal is received and loaded into a computing device, the device can then carry out the various functions described above and herein.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] Fig. 1 is a flowchart illustrating an overall process for reading data from data sources registered with the system to illustrate the invention.

[0014] Fig. 2 is a flowchart illustrating the process for writing data to the data sources registered with the system to illustrate the invention.

[0015] Fig. 3 is a flowchart illustrating a process by the read data manager to illustrate in more detail the functions of the read data manager of Fig. 1.

[0016] Fig. 4 is a flowchart illustrating in more detail the read data manager of Fig. 2.

[0017] Fig. 5 is a flowchart illustrating in more detail the process carried out by the XSL process instruction manager of Fig. 3.

[0018] Fig. 6 is a flowchart illustrating in more detail a process for resolving partial XML node addresses using a service tree.

[0019] Fig. 7A is a flowchart illustrating a process of role based access control

while reading data.

[0020] Fig. 7B is a flowchart illustrating role based access control when writing data to the data sources.

[0021] Fig. 8A is a flowchart illustrating a process by the service lookup manager of Fig. 3 for finding a list of data source or service proxies that would provide data or service requested by the user.

[0022] Fig. 8B is a flowchart illustrating a process by the service lookup manager for adding a data source or service proxies to the registry of the system or adding a service to a registered data source.

[0023] Fig. 8C is a flowchart illustrating a process for removing a data source or service from the system.

[0024] Fig. 8D is a schematic view, where the blocks on the left side of the figure including a Hashmap and Vectors of ServiceWrappers illustrate a first mechanism for mapping an XML node addresses to ServiceWrappers for finding data sources that provide the type of services as indicated by their XML node addresses. Fig. 8D also illustrates a mapping between a hash table of service types and the Vectors of ServiceWrappers on the right side of the figure to illustrate an alternative second embodiment of the mapping mechanism.

[0025] Fig. 8E is a schematic view of an XML tree of XNAs to illustrate the alternative mapping embodiment of the mapping mechanism of Fig. 8D.

[0026] Figs. 9A, 9B together form a flowchart illustrating the functionalities of a dynamic XML generator in collating XML documents into a single XML document to illustrate the invention.

[0027] Figs. 9C-9W are schematic diagrams of XML trees and various markers to illustrate the process by which the XML generator builds an XML tree from XML representations returned by the data sources to illustrate an aspect of the invention.

[0028] Fig. 10 is a flowchart illustrating a process by the HTTP post-data processor for writing data to data sources to illustrate the invention.

[0029] Fig. 11 is a block diagram of a system used for providing data or service from a data source or service to a user to illustrate an embodiment of the invention.

[0030] Fig. 12 is a block functional diagram of a system for a user to write data to a data source or service to illustrate an embodiment of the invention.

[0031] Fig. 13 is a data flow diagram employing an example to illustrate the operation of the various components of the system of this invention.

[0032] Fig. 14 is a flow chart illustrating an XML element role based access control flow system processing a read request from a user role to illustrate one aspect of the invention.

[0033] Fig. 15 is a flow chart illustrating a process by which the XML element role based access control flow system processes a read request from a "User Admin" to illustrate the invention illustrated in Fig. 14.

[0034] Fig. 16 is a block diagram of a computer system useful for illustrating the invention.

[0035] For simplicity in description, identical components are identified by the same numerals in this application.

BRIEF DESCRIPTION OF ONE EMBODIMENT OF THE INVENTION

[0036] The one embodiment of the invention enables interaction with loosely associated disparate data sources in a uniform manner. Each data source may have complete control of its XML data representation; the embodiment is responsible for retrieving and collating the dissimilar data representations into a uniform and optimal representation while maintaining fine-grained request resolution and security associated with each XML element.

[0037] The embodiment of the invention receives requests for data described as XML Node Addresses from a software entity that has taken on a Role. The received requests are then validated using XML Node Role-based Access Control and the accessible XNAs are dynamically resolved to an available matching data source by the Service Lookup Manager using a Service Tree.

[0038] The Service Lookup Manager uses the Service Tree to create a map describing the available XML nodes and the best fit of data sources that can fulfill the request. This map is then used to identify data sources. In this embodiment of the invention, each data source responds with the requested XML nodes. Once the requested XML nodes from the data sources in response to all data requests have been completely received, the Dynamic XML Generator collates and generates a single XML representation of the full set of requested XML nodes.

[0039] The embodiment of the invention returns the dynamically generated representation of the requested set of data to the software entity that originally placed the request.

DETAILED DESCRIPTION OF THE EMBODIMENTS

[0040] Fig. 11 is a functional diagram illustrating a read operation of one embodiment of the system of this invention. As shown in Fig. 11, in order to obtain desired data, a user sends a HTTP get command by means of an application such as an internet browser 1002 through the internet 1004. The request is received by the system 1100 of this invention comprising functional blocks of software entities 1030-1090. System 1100 preferably includes a web server 1110. Server 1110 preferably is equipped to run software such as servlets that translates the request from the user by means of a common mechanism for accessing the data requested from various different data sources.

[0041] In the preferred embodiment, the user request is translated using an XSL file, where the XSL files are stored in server 1110. Thus, XSL style sheets may be prepared in advance and XSL files containing the style sheets are stored in server 1110, each file having a file name. These file names are exposed to internet browser 1002. Therefore, by sending an HTTP get command that refers to one of the XSL files (e.g. the "user" file as shown in Fig. 11), stored in server 1110, server 1110 will, in turn, respond to the request by retrieving process instructions in the style sheet prepared ahead of time in the XSL file referred to in the user request and presenting the process instructions to the read data manager 1120. The read data manager will, in turn, call on the other software entities 1130-1190 to serve the instructions

presented by server 1110. Obviously other ways of translating user request in a uniform manner by system 1100 other than the XSL style sheets prepared ahead of time may be used and are within the scope of the invention. For example, servlets hosted by the server 1110 may be used to parse the URLs from request by the browser to identify the XML node addresses in the URLs.

[0042] The read data manager 1120 identifies the data sources that have the capability to supply the data requested by the user, reads the data from the data sources through a network 1201 and calls on the appropriate ones of software modules 1130-1190 to collect the responses from the different data sources and convert the data collected into an appropriate form. This data is then sent through server 1110 and the internet 1004 to browser 1002 to be read by the user. System 1100 is connected at the back end through network 1201 to a number of data sources, such as database server or proxy 1202 and the email server or proxy 1204. These servers or proxies are connected to network 1201 through interfaces 1206, 1208. Optionally, these servers may be connected to the interfaces through another network 1210. The networks 1201, 1209 may be private networks or a public one such as the internet, or combinations thereof.

[0043] System 1100 enables the user to write data into, or read data from, a number of different disparate data sources without having to have access to particular or specific interfaces adapted for each of the data sources or even knowledge of such interfaces. Thus, system 1100 provides a common mechanism by which users can do so without such capability. In the preferred embodiment, this is made possible by the use of XML node addresses explained in more detail below.

[0044] Thus, in the embodiment where a user request from browser 1002 is translated into an XSL style sheet as described above, the style sheets stored in server 1110 contain XML node addresses of data sources that have the capability to provide the type of information desired by the user, or the appropriate data sources into which data from the user is to be written. In order for the XML node address to be an effective mechanism for accessing the data sources, the data sources (e.g. 1202, 1204) should be pre-registered with system 1100 by informing the system the XML node addresses by which they can be accessed. Once this has been done, XSL files

containing style sheets referring to the XML node addresses may then be written and stored in server 1110. Therefore, when a user requests certain information by requesting a particular XSL file from server 1110, the style sheet of the file will then be presented to the read data manager 1120, and the XML node addresses contained in the style sheet may then be used by system 1100 to identify the appropriate data sources having the capability to supply the information desired by the user. In this manner, the user does not have to have access to interfaces particularly adapted to access the different data sources. Before the operation of system 110 is explained in detail, an overview of the different features of one embodiment of the system may be useful as set out below.

[0045] Fig: 12 is a functional diagram illustrating a write operation of one embodiment of the system of this invention. As shown in Fig. 12, in order to write data to the appropriate data sources, a user sends a HTTP post request by means of internet browser 1002 through the internet 1004. The request is received by the system 1200 of one embodiment of this invention comprising functional blocks of software entities 1210-1260. System 1200 preferably includes a web server 1210, which may be the same as server 1110 of Fig. 11. Server 1210 preferably is equipped to run software such as servlets that translates the request from the user by means of a common mechanism for writing the data from the user to various different data sources, such as sources 1202, 1204.

A. Overview

[0046] XML Node Address (XNA): A data addressing mechanism that enables an individual or collection of XML nodes to be requested from the set of data made available by a data source. The syntax of the address adheres to definition of a URI (see: RFC2396).

[0047] XML Node Role-based Access Control: The facility to determine accessibility of a specific XML Node Address. Since the concept of security at the XML file level does not translate to a dynamically generated XML environment, the concept of XML node based security would be useful. To enable fine-grained security within the dynamically generated XML, the creation of an infrastructure that supports

security down to the individual XML node was preferable. Entities that use a Dynamically Generated XML based system are assigned roles such as "administrator", "user" and/or "guest". Associated with each role is a collection of permissions that allow read, write, add, delete, etc... interaction with an individual XNA or a collection of XNAs available from the collection of disparate data sources. See the section entitled XML Node Role Based Access Control below.

[0048] Dynamic Data Source Resolution: The Data Source Lookup Manager (DSLM) receives the list of XML Node Addresses that are reported by each data source and manages the resolution of addresses to data sources. The DSLM is also referred to herein and in some figures as the Service Lookup Manager (SLM). To successfully resolve XNAs to data sources, the DSLM maintains a mapping of the XML node addresses available to the system 1100 and data sources associated with such XNAs. When a collection of XNAs are simultaneously requested, the DSLM creates a data source request map that describes the available XML nodes and the best fit of data sources that can fulfill the request. If there are redundant data sources within the community, the DSLM uses telemetry information to determine which data source will receive the specific request.

[0049] Data Source Tree: XML DOM describing the relationship between XML Node Addresses and individual data sources. DOM is a widely used term in the industry. For a more complete definition, see www.w3c.org or www.sun.com.

[0050] Data Source: A data source performs two tasks in order to facilitate translation of the source's data into XML and collation of the source's XML data with that of other data sources. First it responds to a get data request, which comes in the form of a XML Node Address. The data source then converts the XNA into an application data specific request, processes that request, and returns an object containing the XNA and the application specific data converted into a small XML Document Object Model (DOM). An individual data source may be asked to process an array of XNAs.

[0051] Dynamic XML Generator: The mechanism for collating and transforming XML nodes received from disparate data sources.

[0052] Dynamically Generated XML: The Community Markup Language is one possible format implementation to describe the dynamically collated XML data.

[0053] Other definitions are set forth below in APPENDIX A attached hereto and made a part of this application.

B. Procedure by Which System 1100 Reads and Writes Data

1. Read Data

[0054] Fig 1. is a flow diagram of reading data due to a client read request. The process starts by receiving HTTP GET request for XSL file (step 110). The request is forwarded to the read data manager 1120 (step 120). Read data manager 1120 is responsible for interacting with the data sources such as sources 1202, 1204 and generating dynamic XML from their responses (step 130). The XML responses are combined and transformed using XSL. The resulting markup is then sent back to the client (step 140).

2. Write Data

[0055] Fig 2. is a flow diagram of writing data due to a client write request. The process starts by receiving HTTP POST request (step 210). The request is forwarded to the write data manager described below in reference to Fig. 12 (step 220). Write data manager is responsible for finding the back end appropriate data sources and writing the data to them (step 230).

3. Read Data Manager 1120

[0056] Fig 3. is a flow diagram of the read data manager 1120. The input consists of a XSL file request. The XSL file is processed by the XSL Process Instruction Manager to extract the XNAs (step 310). The XNAs are then resolved by the Service Tree (step 320), returning an expanded list of XNAs. The requested XNAs are preferably filtered by the XNA RBAC based on the requesting software entity's Role (step 330). The filtered XNAs are sent to the Data Source Lookup Manager, which resolves the corresponding data sources and forwards the requests to them (step 340). The individual XML replies from the individual data sources are collated using the

Dynamic XML Generator (steps 350, 360). The resulting XML document is transformed using XSL to generate the desired output (step 370).

4. Write Data Manager 1220

[0057] Fig 4. is a flow diagram of the write data manager. Write requests are received as HTTP POSTs with name value pairs of the XNAs and data. HTTP Post Data Processor collects XNAs from the POST requests (step 410). All XNAs preferably are sent to the RBAC system (step 420). The XNAs that have been filtered by the RBAC system are sent to the Data Source Lookup Manager, which resolves the corresponding data sources and forwards the write requests to them (steps 430, 440).

5. XML Process Instruction Manager 1130

[0058] Fig 5. is a flow diagram of the XML Process Instruction Manager. This component provides the mechanism for style sheet writers to make XNA calls to the data sources. The XPIM (XML Process Instruction Manager) reads the requested style sheet into a representation in system memory. The XPIM then searches the style sheet for process instructions containing XNAs (steps 510-540). As a final step the XPIM adds the original query parameters (steps 550, 560) to each XNA in the result list that is returned to the read data manager.

[0059] An example of how the XPIM algorithm is described follows. This example is based upon the following information, where machine name is the name of server 1110:

[0060] An HTTP request is made for user number 30's information:

HTTP request: http://<machine_name>/UserInfo.xsl?uid=30

There exists a XSL file named "UserInfo.xsl" that consists of the following XSL.

```
...
<? XML version="1.0" ?>
<xsl:stylesheet>
    <? qs-read user/name/first ?>
    <? qs-read user/name/last ?>
    <? qs-read user/email/address ?>
```

```
<xsl:template match="user">
    ...
</xsl:template>
</xsl:stylesheet>
```

1. Upon receipt of the HTTP request for the UserInfo.xsl file parse UserInfo.xsl searching for process instructions. (Step 510). The user identity information "uid=30" is referred to herein as query parameters.
2. Search for each process instruction (Step 520) containing a XNA requests (Step 530):

   <? qs-read user/name/first ?>

   <? qs-read user/name/last ?>

   <? qs-read user/name/first ?>

   since the following process instruction does not contain an XNA request it will be skipped:

   <? XML version="1.0" ?>
3. Add each found XNA request to the result list. (Step 540)

   user/name/first

   user/name/last

   user/email/address
4. Add "uid=30" from the original request to each XNA request in the result list. (Step 550)
5. Return the result list

   user/name/first?uid=30

   user/name/last?uid=30

   user/email/address?uid=30

6. Data Source Tree 1140

[0061] Fig 6 is a flow diagram of the Data Source Tree. To make it easier for XSL style sheets writers, a truncated form of XNA or partial XNA may be used by the writer. The Data Source Tree of Fig. 6 is used to resolve the truncated or partial XNA to obtain the full XNA or XNAs.

[0062] An example of the partial XNA resolution algorithm used by the Data Source Tree is described as follows. This example is based upon a service tree that has been created in accordance with the following XML hierarchy:

```
<user>
    <name>
        <first/>
        <last/>
    </name>
    <email>
        <address/>
    </email>
</user>
```

For the sake of this example the shorthand "-" character will be used as an indicator of a partial or truncated XNA.

1. Given the partial XNA "user/name/-?uid=30" determine if the XNA is truly partial by searching for the partial XNA shorthand character "-". (Step 610)
2. Remove and store "uid=30" into temporary storage (Step 615)
3. Using the existing Data Source Tree find the partial XNA in the tree as follows (Step 620):
    a) Locate "user" node in Data Source Tree
    b) Locate "name" node underneath "user" node in Data Source Tree
4. There are two children underneath the "name" node. (Step 625)
5. Choose one of the child nodes for further traversal (Step 630)
6. There are no children beneath the two child nodes underneath the "name" node. (Step 635)

7. Add resolved XNA "user/name/first" to the XNA result list. Add resolved XNA "user/name/last" to the XNA result list. (Step 655)
8. There are no more children underneath "user/name" (Step 675)
9. Add the previously stored query parameters "uid=30" to each XNA in the XNA result list. (Step 665)
10. Return result list containing the following XNAs (Step 680):
    "user/name/first?uid=30"
    "user/name/last?uid=30"

In other words, the read data manager identifies a first node in the XML tree that corresponds to a shorthand character in the partial XML node address. It then finds in the XML tree all progeny nodes to the first node; and collects the XML node addresses that correspond to all progeny nodes to the first node.

7. XML Node Address Role Based Access Control 1150

[0063] Fig 7A. is a flow diagram for applying XML Node Address Role Based Access Control to the operation of reading data (step 710). The received XNAs and the Role of the software entity requesting to read the data described by the XNAs are used to determine the permissions for the software entity (steps 720-740). A Role contains a list of Role Permissions that defines a set of XNA permissions. If the Role Permission for the request is not defined, the individual XNA read request is denied. The individual XNAs the Role Permission allows are returned to the Read Data Manager (step 750).

[0064] Fig 7B. is a flow diagram for applying XML Node Address Role Based Access Control to the operation of writing data. The received XNAs and the Role of the software entity requesting to write the data described by the XNAs are used to determine the permissions for the software entity (step 755). A Role contains a list of Role Permissions that defines a set of XNA permissions. If the Role Permission for the request is not defined, the individual XNA write request is denied. Each XNA request is tested to determine if the write permission has been granted (steps 760-780). If any XNA fails the permission test the collective XNA request is denied (step 790).

8. Service Lookup Manager 1160

[0065] Fig 8A. is a flow diagram for reading and writing data requests from the Service Lookup Manager.

1. Strip query parameters off every XML Node Address. (Step 810)
2. While there are XML Node Addresses not mapped to data source or service proxies, where these two terms are used interchangeably herein (Step 811), do the following:
   a) Obtain a list of data source or service proxies that support given XML Node Addresses. (Step 812) This may be implemented by two alternative methods as shown in Figs. 8D, 8E explained below.
   b) For every proxy, check which XML Node Addresses it supports and mark these addresses as mapped. (Step 813)
   c) Within the same data source or service type, get rid of the proxies that support XML Node Addresses fewer than a preset number. (Step 814)
   d) Add a list of data source or service proxies and their supported XML Node Addresses to "best-fit" mapping table. (Step 815)
3. Loop through the entries in the "best-fit" mapping table obtained in steps 811-815 in "best-fit" mapping table and do the following for every entry:
   a) Load-balance between services leaving one data source or service proxy for every data source or service type. (Step 817)
   b) Append previously stripped query parameters to XML Node Addresses. (Step 818)
   c) Add data source or service proxy and its list of supported XML Node Addresses to final mapping table. (Step 819)
4. Return the final mapping table of data source or service proxies and their list of supported XML Node Addresses. (Step 820)

[0066] Fig 8B. is a flow diagram for adding data sources or services to the Data Source Lookup Manager.

1. Get a list of XML Node Addresses from data source or service proxy. (Step 850)
2. Add service ID and service proxy entry into mapping table of data source or service proxies. (Step 851)
3. Make necessary addition updates to Service Tree data collections. (Step 852)

[0067] Fig 8C. is a flow diagram for removing data sources or services from the Data Source Lookup Manager.

1. Remove an entry for service ID of deleted service from mapping table of data source or service proxies. (Step 870)
2. Make necessary deletion updates to Data Source Tree data collections. (Step 871)

[0068] Fig. 8D is a schematic view, where the blocks on the left side of the figure including Hashmap 872 and Vectors 874 of ServiceWrappers illustrate a first mechanism for mapping an XML node addresses to ServiceWrappers for finding data sources that provide the type of services as indicated by their XML node addresses. Fig. 8D also illustrates a mapping between the hash table 882 of service types and the Vectors 874 of ServiceWrappers on the right side of the figure to illustrate an alternative second embodiment of the mapping mechanism. Obviously, not both the combinations indicated above in reference to left and right sides of Fig. 8D are needed; usually only one combination is adequate. Fig. 8E is a schematic view of an XML tree of XNAs to illustrate the alternative mapping embodiment of the mapping mechanism of Fig. 8D.

[0069] In the first embodiment of the mapping mechanism, the uniform resource locators (url 1 through url N) in Hashmap 872 are simply mapped to the corresponding ServiceWrappers in accordance with Fig. 8D. As shown in Fig. 8D, one or more urls may correspond to the same ServiceWrapper. Thus, in step 812 of Fig. 8A, the service lookup manager will look up the urls in the XML node addresses that have not been mapped in the Hashmap 872 and locate the corresponding ServiceWrappers from the mapping shown in Fig. 8D. The ServiceWrappers will, in turn, identify the data source or data sources (e.g. servers 1202, 1204 of Fig. 11) in the

internet cloud that are able to provide the type of service desired.

[0070] Alternatively, according to the second embodiment of the mapping mechanism, in step 812 of Fig. 8A, the service lookup manager may obtain the data sources by means of the XML tree 880 of Fig. 8E and the mapping between hashtable 882 and vectors 874 in the right-hand portion of Fig. 8D. As shown in Fig. 8E, the XML tree 880 comprises a root node and three service type nodes 1-3 are shown. Obiously, more or fewer service type nodes may be employed instead and are within the scope of the invention.

[0071] Each service type node has at least one child node and at least one leaf node of the at least one child node. The service lookup manager in step 812 resolves the tree 880 in view of the unmapped XML node addresses, identifies the service types of the unmapped XML node addresses, uses the service type identified as the key value in hashtable 882 to find the corresponding ServiceWrapper 874 according to the right-hand portion of Fig. 8D. The ServiceWrapper identified, in turn, finds the data source or data sources in the internet cloud of the appropriate service type for providing the services requested by the user via a software entity such as the internet browser1002.

[0072] The service lookup manager identifies the type of service requested by the XML node address. For example, where the XML node address has the root element "user," the service lookup manager would identify user information as the type of service that is requested by the internet browser and identifies the appropriate service type branch of the tree 880 for identifying from the node address the type of service that is requested.

9. Dynamic XML Generator

[0073] The Dynamic XML Generator is used to assemble XML representations or documents given fragments of XML. This is done to facilitate the conversion of XML data from disparate data sources into a single XML representation or document. It is also used throughout the system to convert strings into XML representations or documents. XML representations and XML documents are used interchangeably herein, but with the understanding that XML representations exist only in memory but XML documents may exist in memory or on disk or other external storage devices.

XML DOM is one form of a XML representation.

[0074]

[0075] This component provides the mechanism for collating and transforming partial XML representations or documents received from disparate data sources. DXG (Dynamic XML Generator) has three modes of operation, append, merge, and duplicate deepest node.

a). Append allows for concatenating two XML documents to form the result document.

b). Merge searches the result document for nodes in the partial XML document and places the data found in the partial XML document in the existing leaf in the result document. If the result document does not contain all the nodes in the partial XML document then the remaining partial XML document is inserted in the result document.

c). Duplicate deepest node behaves like Merge mode with the exception that if the deepest node is reached a duplicate node is made in the result tree before the data is added.

[0076] The above described modes allow for combining data from various data sources into a resulting document which can then be understood by XSL transforming engine to produce the desired markup. One of the main advantages of this process is that it provides the XSL writer with a single XML document representing data from multiple sources, as opposed to multiple XML documents, which are extremely difficult to process.

[0077] The XML Generator is primarily used in two places in the system 1100. It is used by the Service Lookup Manager to build a tree, which associates data sources to segments of XML. When an XML segment is requested, the Service Tree allows the system to locate the data source, which can fulfill that request. The data source is contacted, and an XML fragment is returned. The XML Generator is also used by the Read Data Manager 1120 to assemble all of the responses to data source requests into a single document.

[0078] The XML fragments are given to the XML Generator in the form of two strings, one being the SourceXNA, the other being the SourceData. These two strings are converted into a small XML DOM and then attached to another XML DOM, known as the DestinationDOM. Alternatively, the XML Generator can be given SourceDOM in the place of the SourceXNA and SourceData. The SourceDOM is dissected into its component SourceXNAs and SourceDatas, and attached piece by piece to the DestinationDOM. The term "DOM" and "tree" are used interchangeably herein.

[0079] The XML Generator operates in any one of three modes. It can be told to append an XNA and its associated data to the end of a given XML tree. It can completely merge an XNA and its associated data with a given XML tree. It can be told to duplicate the deepest matching node. This works like the merge function, but if the deepest XNA fragment matches a node in the given XML tree, that node will be duplicated and the associated data added within. If only some of the nodes match, the XML Generator will append new nodes based on the unmatched XNA fragments found in the SourceXNA. If no DestinationDOM is supplied, the XML Generator will simply return a DOM consisting of the SourceXNA and SourceData given to it.

[0080] One SourceXNA – SourceData set(s) can be nested within another SourceXNA. This is accomplished by telling the XML Generator to use one SourceXNA as a container for a set of subsequent calls to the XML Generator. After making the call that establishes a given SourceXNA as a container, the XML Generator will return a reference or marker to the place within the XML document being built, where new additions to the XML document are to be appended or merged. Passing this value in as the DestinationDOM, along with the SourceXNA – SourceData sets which are to be contained, will cause those XML document pieces to be placed within the part of the XML document designated as a container in the first call. In short, all additions to an XML document are placed within the part of the document pointed to by DestinationDOM. By moving this reference around, different containers can be established. Thus, as used herein and in an example below, in addition to being the DOM or tree from which tree branches of XML fragments are to be connected or built, DestinationDOM can also refer to the node of the DOM or tree to which pieces of XML fragments are to be placed or attached.

[0081] To avoid XML namespace conflicts, when the XML Generator returns an XML DOM, it insures that the outer most element node (the root node) is called "cml". This is to designate the output of this tree building function as "Community Markup Language", as the type of XML produced by this algorithm.

[0082] These processes are henceforth referred to as XML tree weaving.

Examples of XML Tree Weaving:
Apply the following XML document as the DestinationDOM for the examples to follow.

```
<user>
 <name>
  <first>
   bob
  </first>
  <last>
   Greenland
  </last>
 </name>
 <emailaddress>
  bob.Greenland@mediagate.com
 </emailaddress>
 <greeting>
  Greetings from Sunnyvale California.
 </greeting>
</user>
```

EXAMPLE 1

[0083] In an example of a merge where the deepest node is duplicated before the new data is entered, the XML Generator would be given the following XML fragment to be attached to the tree or DOM in the paragraph immediately above:

SourceXNA user/emailaddress
SourceData bob@yahoo.com

MergeMode DUPLICATE_DEEPEST_NODE

The resulting XML document would look like this:

```
<user>
 <name>
  <first>
   bob
  </first>
  <last>
   Greenland
  </last>
 </name>
 <emailaddress>
  bob.Greenland@mediagate.com
 </emailaddress>
 <emailaddress>
  bob@yahoo.com
 </emailaddress>
 <greeting>
  Greetings from Sunnyvale California.
 </greeting>
</user>
```

[0084] The purpose here is to preserve the separateness of data, in this case the two email addresses. This is needed for proper processing by XSL style sheets and other entities that need to process this document. When the two email addresses are set forth as a list above, one can count the number of email addresses. Otherwise, one would not be able to count the number of email addresses, nor extract one, if they had been fully merged into a single text node.

EXAMPLE 2

[0085] In an example of a merge where the text data is simply added to the deepest matching node, the XML Generator would be given the following XML fragment to be attached to the same tree or DOM:

SourceXNA user/greeting

SourceData Today the NASDAQ hit a record low.

MergeMode MERGE

The resulting XML document would look like this:

```
<user>
 <name>
  <first>
   bob
  </first>
  <last>
   Greenland
  </last>
 </name>
 <emailaddress>
  bob.Greenland@mediagate.com
 </emailaddress>
 <greeting>
   Greetings from Sunnyvale California. Today the NASDAQ hit a
record low.
  </greeting>
</user>
```

[0086] This is done in cases where it is desirable to accumulate text data under a given element node.

EXAMPLE 3

[0087] To perform a simple concatenation of XML data, the XML Generator would be given the following XML fragment to be attached to the same tree or DOM:

SourceXNA user/name/first

SourceData alice

MergeMode APPEND

The resulting XML document would look like this:

```
<user>
 <name>
```

```
<first>
 bob
</first>
<last>
 Greenland
</last>
</name>
<emailaddress>
 bob.Greenland@mediagate.com
</emailaddress>
<greeting>
 Greetings from Sunnyvale California
</greeting>
</user>
<user>
 <name>
  <first>
   alice
  </first>
 </name>
</user>
```

[0088] This is useful when the caller simply wants to append two or more documents together, preserving all data in its original state.

DETAILED EXAMPLE OF FIGURE 9A AND 9B

[0089] The following example demonstrates how data from multiple data sources is aggregated into a single result tree. This is the most common use of the XML Generator.

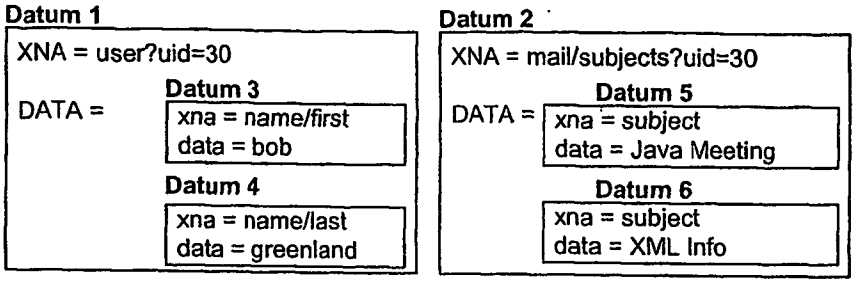
This example requests the following XNAs from the system for user 30:

user/name/first
user/name/last
mail/subjects

[0090] These XNA requests correspond to individual requests for a user's first and last name and a request for a list of email titles belonging to the same user. The purpose is to display a list of message subjects for the given user, along with the user's first and last name on a web page.

[0091] Once these XNAs are requested from the system the actual response received from the corresponding services is an array of objects. For the sake of this example we will call each object within the response array a "datum". A datum describes itself by containing an XNA and the object representing the value associated with the XNA; the datum includes code for calling the XML generator. The object representing the value may be anything, including another datum. Collectively, or individually, each datum can be converted into a specific XML node representation. For the sake of this example the chosen XML representation is an XML DOM (Document Object Model); however, the algorithm contained within the XML Generator may utilize any current or future XML object model.

[0092] The following array of datum describes the response used for the extent of this example:



[0093]

[0094] Each time the XML Generator is called, it will append more data in the form of nodes to the given DestinationTree. The first time it is called, no tree is given and it returns a completely new tree.

A note about the calling sequence used in this example.

[0095] A software entity, such as a servlet, makes a request for data that is to be converted into XML. In an embodiment of the invention, this request is made by the Read Data Manager. The data it requests is the first name, last name, and a list of subject titles for user number 30, in this example. One data source provides the first and last names by sending back a Datum containing the requested information to the software entity that made the request. A second data source provides a list of subject titles in a second Datum.

[0096] After the two response Datum are returned, the software entity (such as the Read Data Manager) which made the request will convert the Datum to XML by calling a method on Datum numbers 1 and 2. When Datum2 is called, the software entity will pass the XML tree, which was the result of Datum1 in as a parameter: This will allow the two XML trees to be attached to one another. The results of both calls will return the root node, indicated by the name "cml" in the example trees.

[0097] There will be a set of 4 intermediate calls made which will build tree fragments. While Datum1 is in the process of building its part of the XML tree, Datum1 will call Datum3 and then Datum4, in much the same way the software entity that made the original request calls Datum1 and Datum2. Each of these calls will cause new nodes to be attached to the XML tree.

Finally Datum1 will return to the calling software entity with half of the completed XML tree. The software entity (such as the Read Data Manager) that made the request will then call Datum2, passing in the XML tree it received from Datum1. Datum2 will then call Datum5 and Datum6, return, and the XML tree will be complete.

Explanation of the terms in the XML fragment passed to the XML Generator.

SourceXNA: An XNA representing a set of nodes to be traversed and or added to the given tree.

SourceData: The text for the text node to be added to the given tree. This value may be null.

IsContainer Flag: The flag describing the fact that additional Source Data items should be added into the tree or DOM at the node pointed to by the reference or marker "DestinationDOM" beneath the current data item. If this flag is TRUE subsequent data items are added as child nodes. If this flag is set to FALSE, subsequent data items are added as peer nodes.

DestinationTree/DestinationDOM: A reference, pointer or marker to a position inside the tree that is being built. It indicates the position at the tree or DOM where new data is to be added. As the XML Generator is often called several times to generate an XML document, this value is often the return value from the previous call.

MergeMode: This can be one of three values. If it is set to *Append*, the XML Generator simply adds new nodes to the position designated by **DestinationTree** without regard to duplicate nodes. If it is set to *Full Merge*, all nodes will be checked for duplicates before appending is done. If set to *Duplicate Deepest Node*, a full merge is performed. Once the tree is traversed, an extra check is performed to see if the last node in the XNA matched the last node traversed in the tree. If this is this case, a duplicate of that node is created and the new text node is added to that node. This allows for the creation of lists. This is the example we will use hereinbelow.

Return reference: The XML Generator will return a reference to the tree being built. If **IsContainer** is set to *false*, the node passed in as **DestinationTree** is returned.

Otherwise, a reference to the last node built is returned so that the next call to XML Generator will build the next part of the tree underneath the returned node.

Step by step execution by the XML generator in reference to Figs. 9A and 9B.

1. The software entity that received the response array calls the XML

Data given to XML Generator

SourceXNA user?uid=30
SourceData NULL
DestinationTree NULL
MergeMode DuplicateDeepest
IsContainer true

Generator with Datum1. The data source that created Datum 1 has set MergeMode to DuplicateDeepestNode, SourceData to "user?uid=30", and SourceData and DestinationTree both to null since there was no tree to begin with. Because Datum1 contains other XML producing entities, the data source or service that created Datum1 has set the IsContainer flag to true.

2. Since the first call was not given a pre-existing tree or DOM or fragments thereof to merge with, a result DOM containing only the CML node is created by the XML generator. (Steps 908 & 909 in Fig. 9A) The result DOM is shown in Fig. 9C.
3. Attributes "uid=30" are stripped from SourceXNA leaving just the value "user". (Step 915)
4. The AppendPoint value is set to the CML node. See Fig. 9D (Step 917)
5. We are not appending two XML documents together. (Step 920)
6. An attempt is made to set MatchRootNode to the first child of the node pointed to by AppendPoint. Since AppendPoint at the CML node has no children, MatchRootNode is set to NULL. (Step 925)
7. Since the MatchRootNode value is NULL and the SourceXNA value is "user" the algorithm skips to step 960. (Step 930)
8. At this point the algorithm has located the place in the source tree to append new data. Because this is the first iteration there has been no tree traversal. The algorithm has created a new DOM consisting

of only the CML node and set AppendPoint to point to the CML node. The algorithm is now ready to add new nodes to the result tree. (Step 960 & 965)

9. The algorithm's mode has been set to DuplicateDeepest and the SourceXNA value is not NULL therefore the algorithm skips to step 977. (Step 970)
10. Since SourceXNA is not NULL strip off all the text up until the first forward slash "/", if any, and create a child node of the same name. The new node will be named "user" and, since there is no trailing slash, the SourceXNA value will be set to NULL. The algorithm adds the "user" node as a child node at the AppendPoint. See Fig. 9E. (Step 980 and 983)
11. The algorithm adds the parameters to the newly created "user" node. See Fig. 9E. (Step 990).
12. The AppendPoint value is set to the "user" node. See Fig. 9F. (Step 985) and loop back to Step 977.
13. Within this iteration, SourceXNA has run out of data and is set to NULL so the algorithm skips to step 987. (Step 977).
14. The SourceData value is NULL so the algorithm does not create a text node. (Step 987).
15. Since the Datum the algorithm is operating upon is a container, the XML Generator returns a reference to the "user" node. This enables the next call by Datum 1 to the XML Generator to append data to the proper location within the result tree. See Fig. 9G. (Step 995).
16. The returned result tree is represented as in Fig. 9F.
17. The XML Generator begins to add Datum3 to the result tree. Because Datum3 simply contains data, the caller (Datum1) tells the XML Generator not to treat it as a container. Because Datum3 is contained by Datum1, the XML Generator is given the node of the current result tree to return to via the DestinationTree parameter.

Data given to XML Generator

SourceXNA name/first **SourceData** bob **DestinationTree** the user node **MergeMode** DuplicateDeepest **IsContainer** false

18. DestinationTree is not NULL and is pointing to the "user" node, so the algorithm skips to step 917. (Steps 910 and 915)
19. AppendPoint is set to the current DestinationTree value. (Step 917).
20. Since the algorithm's MergeMode value is set to DuplicateDeepest the algorithm skips to step 923. (Step 920).
21. The MatchRootNode value is set to NULL because the node to which AppendPoint points to has no children. (Step 925)
22. Since the MatchRootNode value is NULL there is no need to traverse to a non-matching part of the result tree. The SourceXNA value is "name/first" so the algorithm adds new nodes starting at the AppendPoint and skips to step 960. (Step 930)
23. The SourceXNA value is not NULL (Step 977)
24. The algorithm extracts the "name" part of the SourceXNA value and adds a new node to the result tree with a node name of "name". (Step 980).
25. The algorithm removes "name" from the SourceXNA value and sets the AppendPoint value to the newly added "name" node and loops back to step 977. See Fig. 9H. (Steps 983 and 985).
26. The SourceXNA value is still not NULL. (Step 977)
27. The algorithm extracts the "first" part of the SourceXNA value and adds a new child node to the result tree with a node name of "first". Append Point is moved to point to this node. See Fig. 9I. (Step 985)
28. The algorithm removes "first" from the SourceXNA value. Since there the SourceXNA value is empty the SourceXNA value is set to NULL. (Step 983).

29. There are no attributes for the algorithm to add at this time, therefore the "first" node will not be modified. (Step 990). The algorithm again loops back to step 977.
30. This time SourceXNA is null, so the algorithm moves from Seep 977 to Step 987.
31. The SourceData is not NULL therefore the SourceData value is added to the result tree as a text node "bob" beneath the "first" node. See Fig. 9J. (Step 988).
32. Datum3 is not a container; therefore the algorithm returns to Datum 1 a reference to the original DestinationTree value. This means that the algorithm returns a reference to the "user" node as the returned result tree. (Step 997)
33. The XML Generator begins to add Datum4 to the result tree. Because Datum4 simply contains data, the caller (Datum1) tells the

Data given to XML Generator

SourceXNA name/last
SourceData greenland
DestinationDOM the user node
MergeMode DuplicateDeepest
IsContainer false

XML Generator not to treat it as a container. The XML Generator is given the current result tree via the DestinationTree parameter.

34. DestinationTree is not NULL and is pointing to the "user" node, so the algorithm skips to step 917. (Steps 910 and 915).
35. Set AppendPoint to the current DestinationTree value. (Step 917).
36. Since the algorithm's MergeMode value is set to DuplicateDeepest the algorithm skips to step 923. (Step 920).
37. MatchRootNode is not set to NULL because the "user" node has the "name" node as a child node. MatchRootNode now points to the "name" node. See Fig. 9K. The algorithm will now scan the element names in SourceXNA and match them to nodes in the tree until the algorithm ceases to find a matching node name. (Step 925).

38. The MatchRootNode name is the same value as the first part of the SourceXNA value. (Step 940)
39. The node name does match the most recent XNA fragment within the SourceXNA value and this XNA fragment is not the last, so that the algorithm will therefore progress to step 927. (Step 945
40. The algorithm will now trim the "name" XNA fragment from the SourceXNA value and move the MatchRootNode value down to the "first" node. See Fig. 9L. (Step 927)
41. The MatchRootNode value is not NULL and the SourceXNA value still contains data. (Step 930)
42. The name of the tree element pointed to by MatchRootNode does not match the XNA fragment contained within the SourceXNA value. The MatchRootNode value is pointing to the "first" node and XNA fragment contained within the SourceXNA value is "last". (Steps 935 & 940)
43. Since the "first" node is the only child of the "name" node, MatchRootNode has no siblings for the algorithm to test (Step 950)
44. Since the "first" node did not match the XNA fragment value "last" the algorithm sets the MatchRootNode value to the last known matching node. The "name" node is the last matching node and the MatchRootnode value is now set to point to this node. See Fig. 9M. (Step 959)
45. There are no more possible matches and SourceXNA still has a remaining XNA fragment to be added to the result tree, so the algorithm will begin appending new nodes at the current MatchRootNode. (Step 930)
46. The algorithm now sets the AppendPoint to point to the same node as MatchRootNode. See Fig. 9N. (Step 965)
47. Since the algorithm's MergeMode value is set to DuplicateDeepest but SourceXNA is not NULL the algorithm skips to step 977. (Step 970 and 973)
48. The SourceXNA value is not NULL, so the algorithm moves to step 980. (Step 977)

49. The algorithm extracts the "last" part of the SourceXNA value and adds a new node to the result tree with a node name of "last". See Fig. 9O. (Step 980)
50. There are no parameters to add to the "last" node, therefore the algorithm simply moves the AppendPoint to the node named "last" and sets the SourceXNA to NULL. The algorithm now loops back to step 977.
51. Because the SourceXNA value is NULL the algorithm will add SourceData to the AppendPoint as a text node. See Fig. 9P. (Step987, 988)
52. Datum4 is not a container therefore the algorithm returns a reference to the original DestinationTree value. This means that the algorithm returns a reference to the "user" node as the returned result tree. (Step 997)
53. The result tree describing Datum1 is now completely built. The software entity that is iterating over the response array now receives a result to its initial call to the XML Generator. This result is a reference to the "CML" node within the current result tree. If Datum1 had been the only requested data the example would be finished at this point. The final tree for Datum1 is represented in Fig. 9Q.
54. Since there are additional elements in the initial received response array, the XML Generator is now called by the Read Data Manager to build the Datum2 portion of the tree. This process is similar as previously described for the call that built the Datum1 portion of the result tree. The only difference is that the current result tree is passed in as the DestinationTree. The call looks like this:

Data given to XML Generator

SourceXNA	mail/subjects
SourceData	null
DestinationDOM	the cml node
MergeMode	DuplicateDeepest
IsContainer	true

55. Following the same process as described for Datum1 the new result tree is returned that describes both Datum1 and Datum2 within one tree hierarchy. Since Datum2 is a container, the reference is set to the deepest node created, as it was for Datum1.
56. Datum5 is built the same way Datum3 and Datum4 were. See Fig. 9R. Datum3 and Datum 4 are not containers, so the returned reference to the result tree points to the node named "subjects". See Fig. 9S.
57. Datum6 contains an XNA that is exactly the same as Datum5. If the algorithm was configured with a MergeMode of "Full Merge", an additional text node would be inserted below the existing result tree's "subject" node. When processed, there would appear to be only a single electronic mail subject, titled "Java Meeting XML Info". The desired result is instead to produce a list of "subject" nodes under the "subjects" node; therefore, the deepest node must be duplicated if it matches the last part of SourceXNA value.
58. The final call to the XML Generator looks like this:

Data given to XML Generator

SourceXNA	subject
SourceData	XML Info
DestinationDOM	the subjects node
MergeMode	DuplicateDeepest
IsContainer	false

59. For the first time, the SourceXNA is a complete match. (Step 927) This causes the algorithm to move the AppendPoint directly to the "subject" node and MatchRootNode to the "Java Meeting" node. See Fig. 9T. The SourceXNA value is set to NULL. The algorithm performs an additional check to determine if the attributes given in the SourceXNA value match those of the existing "subject" node. Since the attributes associated with the SourceXNA and the

"subject" node are both NULL the attributes are determined to match the algorithm now progresses to step 970 through steps 930 and 959, which causes Match RootNode to back up to "subject." (Step 953)

60. The SourceXNA value is NULL since a complete match had taken place. Since the MergeMode is "DuplicateDeepest we must prevent the "Java Meeting" text from being merged with the "XML Info" text. To accomplish this, the algorithm moves the AppendPoint up one level from Match RootNode and sets SourceXNA value to "subjects". See Fig. 9U. (Steps 965, 973 & 975)
61. The XML Generator will operate as it did when it built Datum3, Datum4, and Datum5. The XML Generator will add an additional "subject" node. The second "subject" node will contain a text node containing the value of the SourceData, "XML Info". (Steps 980, 988)
62. The XML Generator will return a reference to the "subjects" node. See Fig. 9V.
63. Finally the call, which produced the XML representation of Datum2, returns with a reference to the root, and the result tree is complete. See Fig. 9W.

10. HTTP Post Data Processor

[0098] The HTTP Post Data Processor 1240 is used to convert data supplied by HTML forms into data that can be submitted into our system. It allows the developer to construct generic HTTP forms, which can be used to convey data to the system.

[0099] This component accomplishes this by providing a mechanism for searching for XNA'S in the name value pairs that are sent in the HTTP Post request. HPDP (HTTP Post Data Processor) is designed to look for names in the request containing the XNA Prefix and collect them into a list. XNA Prefix is a predefined string that allows the user to specify XNAs in existing markup language input tags. The remaining names are concatenated to form the query parameters to each XNA in the resulting list.

[00100] The HTTP Post Data Processor 1240 is used by the HTTPFormAdapter software entity which may be hosted by server 1210.

[00101] When a HTTP POST request comes into a web server 1210 of Fig. 12 hosting a HTTPFormAdapter, this adapter makes a call to the HTTP Post Data Processor, passing it the set of name-value pairs it received. (Step 1010 of Fig. 10)

[00102] The HTTP Post Data Processor first scans through these name-value pairs, looking for all names that do not begin with a Write Data Prefix. It assembles all of these into a list, which will later be used as parameters to data source calls. (Step 1020)

[00103] The Processor then scans the list again, looking for all name value pairs that begin with a Write Data Prefix. It extracts the XNA from the name containing a Write Data Prefix and stores these XNAs in a list, appending the parameter list collected in the first scan to each extracted XNA, and associating the value with the newly constructed XNA. (Step 1030-1060)

[00104] Finally, the Processor sends the XNA – value sets into the system in the form of WriteData requests. (Step 1070) Using the Data Source or Service Lookup Manager 1250, the system locates the data source associated with each XNA, and instructs that data source to set the new value to the value prescribed by the HTTP Post Data Processor.

Example of HTTP Post Data Processing

[00105] In this example the user associated with the unique id "uid" value of 30 will have their first name set to "bob", and their last name set to "greenland".

If given the following HTML Form:

```
<form action="HTTPFormAdapter?uid=30" method="POST">
<input type="text" name="qs-write-data user/name/first"
value="bob"/>
<input type="text" name="qs-write-data user/name/last"
value="greenland"/>
<input type="submit"/>
</form>
```

[00106] When the user clicks on the submit button, a name-value pair list is sent back to the HTTP server. The HTTPFormAdapter receives the name-value pair list and sends them off to be processed. The name value pairs would look like this:

qs-write-data user/name/first = bob
qs-write-data user/name/last = greenland
uid=30

[00107] The HTTP Post Data Processor first scans through the list, looking for all names that do not being with a Write Data Prefix. The Processor finds the following.

uid=30

[00108] Next, the Processor scans through the name value pair list looking for all names that begin with a Write Data Prefix and finds the following:

qs-write-data user/name/first = bob
qs-write-data user/name/last = greenland

[00109] As the HTTP Post Data Processor scans the list, the Write Data prefix is stripped off, leaving only the XNA and its associated value.

user/name/first = bob
user/name/last = greenland

[00110] The parameter list, which was processed in the first pass, is added to each of the XNAs found in the second pass. The list now consists of the following.

user/name/first?uid=30 = bob
user/name/last?uid=30 = greenland

[00111] Finally, using the Data Source or Service Lookup Manager 1250 the data sources associated with each XNA are located and the write requests are sent to the proper data sources.

[00112] Fig. 13. labeled XML Generator and Process Instruction Manager

Example summarizes the result of these algorithms as explained above.

Role Based Access Control

[00113] Described below are one embodiment of an apparatus and method for Role Based Access Control to individual XML (extended mark-up language) Elements within a static or dynamically generated XML object representation.

[00114] The embodiment of the invention uses Access Control Logic to grant access to XML Elements within a static or dynamically generated XML object representation. The mechanism provides for any view of an XML object representation on a per user basis via a *permission*. A *permission* can grant access to a single XML Element or a cluster of XML Elements. Access can be restricted to *read-only*, *write*, *add*, *delete*, *execute* or *none*. A view is referred to as a *Role*. A *Role* consists of one or more *permissions* granting or restricting access to XML Elements. One or more *Roles* can exist simultaneously within a single operating environment to provide as many unique views of the XML object representation as desired.

The Roles

[00115] Roles provide Access Control to the XML Elements of an XML object representation. For the sake of this discussion the object representation that will be used is a XML DOM. A *Role* enables permission to view specific elements within a XML DOM. Multiple *Roles* provide multiple permissions to view a XML DOM. A *Role* can be defined for an individual, perhaps a System Administrator. This *Role* might grant a larger collection of access permissions to a XML DOM than a *Role* defined for non-administrative individual, perhaps a User. As many *Roles* can exist in the system as desired. Each *Role* is a collection of one or more *permissions* as discussed below.

[00116] Individual *Roles* should be thought of as filters for unique views into an XML object representation. The implementation is such that a XML DOM is dynamically generated according to the *Role* associated with the *request*. A *request* is a *read, write, add, delete or execute* of a specific XML Element. Applying the *Role* to the *request* prior to the retrieval of the data from the data source will increase the performance of the software entity leveraging this invention based on the filtering

capabilities. If a *Role* does not have the proper access for the *request*, only the subset of XML Elements that satisfy the *Role* will be processed. The end result is an XML DOM that contains only the data granted by a *Role*. This mechanism is what guarantees the secure creation of a XML object representation.

The Permissions

[00117] A *permission* describes a specific view of an XML object representation as well as the access type for the XML Element node or Element nodes. The following defines the nomenclature of a XNAPermission -

[00118] XNAPermission <XML Element(s)>,<Access Type>;

[00119] The first component of the *permission* is the XML Element. An element separator is the forward slash character (/). The root of the XML object representation is implied, thus there is not a forward slash preceding the description for the XML Element. There are special tokens that can accompany the XML Element that allow a *permission* to span more than one XML Element. The tokens are a star character (*) representing a wildcard, a dash character (-) representing any XML Element recursively under the given XML Element and a plus character (+) representing the XML Element and any XML Element recursively under the given XML Element. These are easily described by the following examples:

1. XNAPermission "users/data/*","read"; grants read access to all XML Elements one level deep under the "users/data" element. This does not include the "users/data" element.

2. XNAPermission "users/data/-","read"; grants read access to all XML Elements recursively under the "users/data" element. This does not include the "users/data" element.

3. XNAPermission "users/data/+","read"; grants read access to all XML Elements recursively under the "users/data" element including the "users/data" element.

4. XNAPermission "*/*/data","read"; grants read access to <any_element>/<any_element>/data. For example, users/test1/data, groups/test2/data, admin/test3/data etc.. The star (*) can be used in any element location to indicate a wildcard.

5. XNAPermission "*/*/d*t*","read"; grants read access to <any_element>/<any_element>/d<any characters>t<any characters>. For example, users/test1/determine, groups/test2/dayton, admin/test3/duty etc.. The star (*) can be used in any location and as many times as necessary within the XNA to indicate a wildcard condition.

6. XNAPermission "-","read"; as well as XNAPermission "+","read"; grant read access recursively to all XML Elements within the XML object representation.

The second component of the *permission* is the Access Type granted. The Access Types are:

- "none" - grants no access
- "read" - grants read access
- "write" - grants write access
- "add" - grants add access
- "delete" - grants delete access
- "execute" - grants execute access

The Access Type keyword implies an order of preference as follows:

1. "none" (cannot be used in conjunction with other Access Types in the *permission*)

2. "read" (as the only Access Type in the *permission*)

3. "read", "write", "add","delete" or "execute"

[00120] This means that if a *permission* Access Type is specified as "none" and the *permissions* XML Element overlaps a *permission* that is granted "read" access within the same *Role*, there will be no access to the XML Element.

[00121] Fig. 14 is a flow chart illustrating a XML element role based access control flow system processing a read request from a user role to illustrate one aspect of the invention. Prior to the operation as indicated in Fig. 14, the permissions that are granted for the various roles, including the "User" role has already been installed in software entity 1150. Thus, using the terminology defined above, the permission that is granted to a "user" role and stored in memory accessible to software entity 1150 may be defined as follows:

XNA PermissionM "User/-","Read";

XNA Permission P "User/application","None".

[00122] The first line of the XNA Permission indicates that the requester has permission to obtain data from all data sources whose XML node addresses starts which the term "User," The second line of the XNA Permission indicates that the user does not have permission to access any applications from the data sources. Thus, taken together, the permssion granted to this particular role is to be able to obtain data from data sources whose XML node addresses starts which the term "User," except that it does not have any permission to access any applications from these data sources.

[00123] As shown in Fig. 14, the user submits the following read request 1302 to system 1100 of Fig. 11:

ReadRequest by a "User":

<user.

<first/>

<last/>

<application/>

</user>

[00124] The role based access control software entity 1150 of Fig. 11 applies a filter based on the "User" role (step 1304) defined above so that software entity 1150 will be able to compare the request submitted by the user in step 1302 to the stored permissions for the particular role involved. Thus, as noted above, the "User" role has permission to obtain data from data sources whose XML node addresses starts which the term "User," except that it does not have any permission to access any applications.

[00125] Software entity 1150 therefore filters the XML node address submitted by the user to remove any reference to applications and submits the filtered XML node address to the read data manager 1120 of Fig. 11 as indicated in step 1306. The read data manager handles the request in a manner as described above (step 1308) and returns the result in the manner described above to the user through browser 1002 (step 1310).

[00126] As opposed to the "User" role, an administrator may have access to all data sources with XML node addresses that start with the term "User." Again the permission for this role may be defined as follows and is stored in memory associated with software entity 1150:

UserAdmin role defined as:

XNA PermissionM "User/-","Read";

[00127] The operation of software entity 1150 is illustrated in Fig. 15. In reference to Fig. 15, again browser 1102 submits a read request by a "UserAdmin," where the read request is exactly the same as that submitted by the user in step 1302 (step 1322). Software entity 1150 again compares the read request to the permission stored therein for the role "UserAdmin" and finds that the entire request may be granted so that the entire request is passed to the read data manager (steps 1324, 1326). The read data manager then obtains the data and returns the data to browser 1102 in the manner described above (steps 1328, 1330).

[00128] While the invention has been described above by reference to methods and data structures, it will be understood that these methods and data structures may be stored in media and transported, so that they can be installed into any intelligent or computing device and used on the device. The methods and data structures can also be embodied in signals transmitted through any channel, including wired, wireless and optical channels.

[00129] Fig. 16 shows an information appliance (or digital device) that may be understood as a logical apparatus that can read instructions from media 1417 and/or network port 419 in order to install software embodying the above-described methods and data structures. Apparatus 1400 can thereafter use these methods and data structures to direct server or client logic, as understood in the art, to embody aspects of the invention. One type of logical apparatus that may embody the invention is a computer system as illustrated in 1400, containing CPU 1404, optional input devices 1409 and 1411, disk drives 1415 and optional monitor 1405. Fixed media 1417 may be used to program such a system and may represent a disk-type optical or magnetic media, magnetic tape, solid state memory, etc.. One or more aspects of the invention may be embodied in whole or in part as software recorded on this fixed media. Communication port 1419 may also be used to initially receive signals carrying instructions that are used to program such a system to perform any one or more of the above-described functions and may represent any type of communication connection, such as to the internet or any other computer network. The instructions or program may be transmitted directly to a client's device or be placed on a network, such as a website of the internet to be accessible through a client's device. All such methods of making the program or software component available to clients are known to those in the art and will not be described here.

[00130] The invention also may be embodied in whole or in part within the circuitry of an application specific integrated circuit (ASIC) or a programmable logic device (PLD). In such a case, the invention may be embodied in a computer understandable descriptor language which may be used to create an ASIC or PLD that operates as herein described.

[00131] While the invention has been described above by reference to various

embodiments, it will be understood that changes and modifications may be made without departing from the scope of the invention, which is to be defined only by the appended claims and their equivalents. All references referred to herein are incorporated in their entireties by reference.

APPENDIX A

GLOSSARY OF TERMS

SourceXNA:

The variable called SourceXNA is used in the description of the Dynamic XML Generator. It is an XNA that is given to the XML Generator so that it can be converted into an XML Fragment and added into the DistinationDOM. In our implementation, the SourceXNA is in the form of a Java String. Combined with SourceData, the XML Generator can construct a complete XML fragment.

Example:

SourceXNA User/first?uid=30 SourceData bob

The resulting XML fragment

```
<user>
 <first uid=30>
  bob
 </first>
</user>
```

SourceData:

The variable called SourceData is used in the description of the Dynamic XML Generator. It is a string representation of the text node that is to be placed at the leaf of the XML fragment, which is to be added to the given DistinationDOM.

XNA Fragment:

A single part of a full XNA. If an XNA consists of user/name/first the fragments that make up the XNA are user, name, and first.

DestinationDOM

The variable called DestinationDOM is used in the description of the Dynamic XML Generator. It is a reference to the XML tree which is to be changed by the XML Generator.

AppendPoint

The location in the interim result DOM that newly created DOM pieces are added.

DOM (Document Object Model)

This is a representation of an XML document in a computer's memory. DOM is a widely used term in the industry. For a more complete definition, see www.w3c.org or www.sun.com.

Element Node

An Element Node is a node in an XML document (or DOM), which is surrounded by greater then and less than signs, "<" and ">".

Example:

```
<name>
 bob
</name>
```

The node "name" is an element node. It contains the text node "bob". There is only a single element node in this example; the second (closing) node simply indicates that it is a container.

Text Node

A piece of data found at the leaves of XML documents. This appears as free text set between opening and closing element nodes. *See element node example.*

XNA (XML Node Address)

This is a string representation of a node within an XML document.

Example:

```
<cml>
 <user>
  <name>
   bob
  </name>
 </user>
```

</cml>

An XNA which points to "bob" would be "cml/user/name".

NAME VALUE PAIRS

This is a commonly used term in the software industry. Here we use the phrase in the description of the HTTP Post Data Processor. A Name Value Pair is a set of string data that is associated to one another. When an HTTP Post is performed, all the data from the HTTP Form is sent to the server as Name Value Pairs.

Example:

If given the following HTTP Form

<form action="HTTPFormAdapter?uid=30" method="POST">
<input type="text" name="qs-write-data user/name/first"
value="bob"/>
</form>

The name value pairs received by the HTTP server would look like this.

qs-write-data user/name/first = bob
uid=30

HTTPFORMADAPTER

This is a software module that processes HTML forms.

WRITE DATA PREFIX

The Write Data Prefix is a special name that can be introduced into an input tag of an HTML form which signals the HTTP Post Processor that the associated piece of data is to be submitted to the system.

In the Name Value Pairs example, the "qs-write-data" portion of the name in the form is a Write Data Prefix. It instructs the system to send the value "bob" to the data source that is known to the system as "users/name/first". The additional parameter "uid=30" is also passed to the data source whose address is "users/name/first", therefore the resulting XNA would be "users/name/first?uid=30".

WHAT IS CLAIMED IS:

1. A method for obtaining information from a group of data sources that supply data in response to a request by a software entity, said sources having a common XML-related addressing mechanism, comprising:

reading a request by a software entity;

providing from among the group a set of one or more data source(s) that are capable of supplying data in response to the request by means of the common mechanism; and

obtaining from at least one of the data source(s) provided data requested by the software entity.

2. The method of claim 1, wherein said common mechanism comprises XML node addresses and wherein said reading derives the XML node addresses from the software entity request.

3. The method of claim 2, wherein said common mechanism also comprises a vehicle for obtaining from the XML node addresses the set of data source(s), and wherein said providing provides the set of data source(s) by means of the vehicle and the XML node addresses read from the software entity request.

4. The method of claim 3, wherein said vehicle comprises a service tree for finding the set of data source(s) from XML node addresses, and wherein said providing comprises resolving the service tree using the XML node addresses from the software entity request to find the set of data source(s).

5. The method of claim 3, wherein said vehicle comprises a table useful for providing data source(s) from the XML node addresses, wherein said providing provides from the table two or more data sources that correspond to a XML node address read from the software entity request, said method further comprising selecting said at least one data source from the two or more data sources obtained from the table.

6. The method of claim 3, wherein said vehicle comprises a hash table mapping data sources to the XML node addresses, wherein said providing provides from the table two or more data sources that correspond to a XML node address read from the software entity request.

7. The method of claim 1, wherein said obtaining obtains the data from a fewer number of data source(s) than those data sources identified to be capable of servicing the request, based on loads on the identified plurality of data sources.

8. The method of claim 1, further comprising discarding at least one of the data source(s) identified to be capable of servicing the request when such source supports fewer than a predetermined number of services serving process instructions, wherein said obtaining obtains data from data source(s) that remain after the discard.

9. The method of claim 1, said reading comprising translating the request by a software entity into process instruction(s) using a XSL file.

10. The method of claim 9, wherein said common mechanism comprises XML node addresses and wherein said process instruction(s) comprises XML node addresses.

11. The method of claim 9, wherein said translating supplies XSL files containing process instruction(s), and the reading obtains the process instruction(s) by identifying a XSL file name from the request by the software entity.

12. A method for obtaining information from a group of data sources, comprising:

reading a request from a software entity;

identifying data sources among the group that are capable of servicing the request;

obtaining, from a plurality of the identified data sources, source XML representations in response to the request; and

collating the source XML representations into a single XML representation, wherein said collating comprises matching nodes in the source XML representations and

13. The method of claim 12, wherein said source XML representations have matching deepest nodes, and said collating comprises merging data at deepest matching nodes of the XML documents.

14. The method of claim 13, wherein data from different source XML representations are merged at different deepest nodes.

15. The method of claim 12, wherein data from different source XML representations are merged at the same deepest node(s).

16. The method of claim 12, at least one of said source XML representations indicating format for data presentation, wherein said collating presents data from the at least one of said source XML representations according to the format indicated.

17. The method of claim 16, at least one of said source XML representations indicating a hierarchical format for data presentation, wherein said collating presents data from the at least one of said source XML representations in a tree according to the hierarchy of the format indicated, so that a data source of the at least one of said source XML representations has complete control of presentation of data from such data source.

18. The method of claim 12, each of at least some of said source XML representations indicating format for data presentation as source trees comprising nodes, wherein said collating presents data from the at least some of said source XML representations in a result tree that comprises the source trees as branches.

19. The method of claim 18, wherein said collating comprises building the result tree using markers to indicate nodes where one or more branches are to be formed.

20. The method of claim 19, wherein said collating builds a first and a second source tree, each corresponding to one of two of the source XML representations, and uses a marker to indicate a node at a first source tree where the second source tree is to be connected as a branch.

21. The method of claim 18, at least one of said source XML representations comprising a XML node address and at least one object representing a value associated with the address, wherein said collating comprises building a source tree using marker(s) to indicate nodes where branches of the source tree are to be built according to the address.

22. A method for obtaining information from a group of data sources, comprising:

receiving a request from a software entity;

identifying data sources among the group that are capable of servicing the request;

obtaining, from a plurality of the identified data sources, source XML representations in response to the request, wherein at least one of the source XML representations comprises one or more XML node addresses; and

collating the source XML representations into a single XML representation, wherein said collating comprises resolving the one or more XML node addresses. and

23. The method of claim 22, wherein said collating builds one or more source trees comprising nodes when resolving the one or more XML node addresses. and

24. The method of claim 23, said at least one of said source XML representations comprising a XML node address and at least one object representing a value associated with the address, said at least one object comprising at least one XML node address, wherein said collating comprises building the source trees by resolving the XML node addresses.

25. The method of claim 22, wherein said collating employs marker(s) to indicate the node(s) where the source trees are connected to form a result tree.

26. An interface in a computer system for transfer of information to and/or from a group of data sources that supply data to serve request(s) by a software entity, said interface comprising a common addressing mechanism for the group of data sources, said mechanism comprising XML hierarchical data structures and a common method for accessing the structures.

27. The interface of claim 26, said XML hierarchical data structures comprising XML node addresses.

28. The interface of claim 27, at least one of said XML node addresses comprising one or more nodes.

29. The interface of claim 27, at least one of said XML node addresses comprising a shorthand character and/or a wildcard character.

30. A XML mapping mechanism comprising a service tree, said service tree comprising:

one or more service type nodes;

at least one additional node that is a child of one of said one or more service type nodes; and

one or more leaf nodes that are children of the at least one additional node.

31. The mechanism of claim 30, further comprising a hash table listing service type(s) of the tree as the key(s), and locator(s) for data sources that provide service(s) of the type(s) of the corresponding key(s).

32. A method for finding data sources that provide services requested by a software entity by means of a mapping mechanism comprising a service tree, comprising:

parsing a XML node address submitted by the software entity;

finding a match in the service tree to the XML node address; and

locating at least one data source that provides services requested by the software entity.

33. The method of claim 32, said finding comprising identifying type of service requested by the software entity.

34. The method of claim 33, said locating comprising looking up a hashtable to find a ServiceWrapper that locates the at least one data source.

35. A method for access control, comprising:

receiving a request of a software entity for access to one or more data sources, said request comprising a XML node address, said XML node address being used for accessing the data sources;

finding XML element nodes in the XML node address;

comparing the XML element nodes found to XML element nodes for which the software entity has permission to access; and

granting software entity access to the data source(s) with respect to those XML elements for which the software entity has permission to access.

36. The method of claim 35, wherein said XML element nodes in the XML node address comprises shorthand character and/or a wildcard character, and/or a recursive indicator.

37. The method of claim 35, further comprising obtaining from the data source(s) only information for which the software entity has permission to access and returning such information to the software entity.

38. A method for resolving a partial XML node address by means of a XML tree, comprising:

identifying a first node in the XML tree that corresponds to a shorthand character in the partial XML node address;

finding in the XML tree all progeny nodes to the first node; and

collecting the XML node addresses that correspond to all progeny nodes to the first node.

39. A method for writing information to data sources, in response to a request by a software entity, said sources having a common XML-related addressing mechanism, comprising:

reading information concerning a set of one or more addresses of data sources and values corresponding to the address(es) to be written to such data source(s) from a request sent by a software entity;

identifying a set of one or more data source(s) that have the one or more addresses by means of the common mechanism; and

writing to the set of one or more data source(s) the values read from the request by the software entity.

40. The method of claim 39, wherein said common mechanism comprises XML node addresses and wherein said reading derives XML node addresses from the software entity request.

41. The method of claim 40, wherein the request supplies name-value pairs, and the reading parses the pairs to provide XML addresses-value sets.

42. A computer readable storage device storing a program of instructions executable by a computer to perform a method for obtaining information from a group of data sources that supply data in response to a request by a software entity, said sources having a common XML-related addressing mechanism, said method comprising:

reading a request by a software entity;

providing from among the group a set of one or more data source(s) that are capable of supplying data in response to the request by means of the common mechanism; and

obtaining from at least one of the data source(s) provided data requested by the software entity.

43. A signal carrying information on a program of instructions executable by a computer to perform a method for obtaining information from a group of data sources that supply data in response to a request by a software entity, said sources having a common XML-related addressing mechanism, said method comprising:

reading a request by a software entity;

providing from among the group a set of one or more data source(s) that are capable of supplying data in response to the request by means of the common mechanism; and

obtaining from at least one of the data source(s) provided data requested by the software entity.

44. A method for transmitting a program of instructions executable by a computer to perform a method for obtaining information from a group of data sources that supply data in response to a request by a software entity, said sources having a common XML-related addressing mechanism, said method comprising:

causing a program of instructions to be transmitted to a client device, thereby enabling the client device to perform, by means of such program, the following process:

reading a request by a software entity;

providing from among the group a set of one or more data source(s) that are capable of supplying data in response to the request by means of the common mechanism; and

obtaining from at least one of the data source(s) provided data requested by the software entity.

45. A computer readable storage device storing a program of instructions executable by a computer to perform a method for obtaining information from a group of data sources, said method comprising:

reading a request from a software entity;

identifying data sources among the group that are capable of servicing the request;

obtaining, from a plurality of the identified data sources, source XML representations in response to the request; and

collating the source XML representations into a single XML representation, wherein said collating comprises matching nodes in the source XML representations. and

46. A signal carrying information on a program of instructions executable by a computer to perform a method for obtaining information from a group of data sources, said method comprising:

reading a request from a software entity;

identifying data sources among the group that are capable of servicing the request;

obtaining, from a plurality of the identified data sources, source XML representations in response to the request; and

collating the source XML representations into a single XML representation, wherein said collating comprises matching nodes in the source XML representations. and

47. A computer readable storage device storing a program of instructions executable by a computer to perform a method for obtaining information from a group of data sources, said method comprising:

receiving a request from a software entity;

identifying data sources among the group that are capable of servicing the request;

obtaining, from a plurality of the identified data sources, source XML representations in response to the request, wherein at least one of the source XML representations comprises one or more XML node addresses; and

collating the source XML representations into a single XML representation, wherein said collating comprises resolving the one or more XML node addresses. and

48. A signal carrying information on a program of instructions executable by a computer to perform a method for obtaining information from a group of data sources, said method comprising:

receiving a request from a software entity;

identifying data sources among the group that are capable of servicing the request;

obtaining, from a plurality of the identified data sources, source XML representations in response to the request, wherein at least one of the source XML representations comprises one or more XML node addresses; and

collating the source XML representations into a single XML representation, wherein said collating comprises resolving the one or more XML node addresses and

49. A computer readable storage device storing a program of instructions executable by a computer to perform a method for finding data sources that provide services requested by a software entity by means of a mapping mechanism comprising a service tree, said method comprising:

parsing a XML node address submitted by the software entity;

finding a match in the service tree to the XML node address; and

locating at least one data source that provides services requested by the software entity.

50. A signal carrying information on a program of instructions executable by a computer to perform a method for finding data sources that provide services requested by a software entity by means of a mapping mechanism comprising a service tree, said method comprising:

parsing a XML node address submitted by the software entity;

finding a match in the service tree to the XML node address; and

locating at least one data source that provides services requested by the software entity.

51. A computer readable storage device storing a program of instructions executable by a computer to perform a method for access control, said method comprising:

receiving a request of a software entity for access to one or more data sources, said request comprising a XML node address, said XML node address being used for accessing the data sources;

finding XML element nodes in the XML node address;

comparing the XML element nodes found to XML element nodes for which the software entity has permission to access; and

granting software entity access to the data source(s) with respect to those XML elements for which the software entity has permission to access.

52. A signal carrying information on a program of instructions executable by a computer to perform a method for access control, said method comprising:

receiving a request of a software entity for access to one or more data sources, said request comprising a XML node address, said XML node address being used for accessing the data sources;

finding XML element nodes in the XML node address;

comparing the XML element nodes found to XML element nodes for which the software entity has permission to access; and

granting software entity access to the data source(s) with respect to those XML elements for which the software entity has permission to access.

53. A computer readable storage device storing a program of instructions executable by a computer to perform a method for resolving a partial XML node address by means of a XML tree, said method comprising:

identifying a first node in the XML tree that corresponds to a shorthand character in the partial XML node address;

finding in the XML tree all progeny nodes to the first node; and

collecting the XML node addresses that correspond to all progeny nodes to the first node.

54. A signal carrying information on a program of instructions executable by a computer to perform a method for resolving a partial XML node address by means of a XML tree, said method comprising:

identifying a first node in the XML tree that corresponds to a shorthand character in the partial XML node address;

finding in the XML tree all progeny nodes to the first node; and

collecting the XML node addresses that correspond to all progeny nodes to the first node.

Read Data

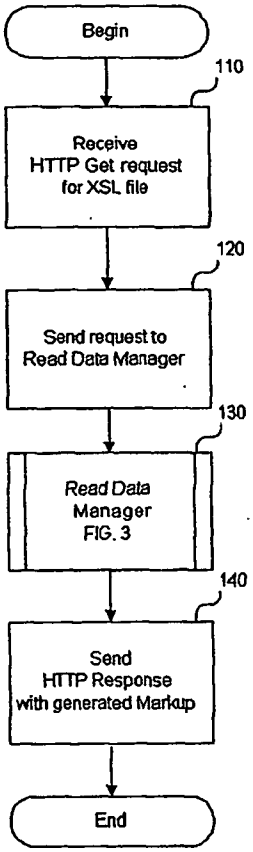


FIG. 1

Write Data

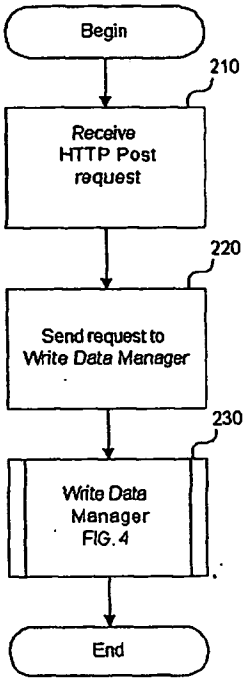


FIG. 2

Read Data Manager

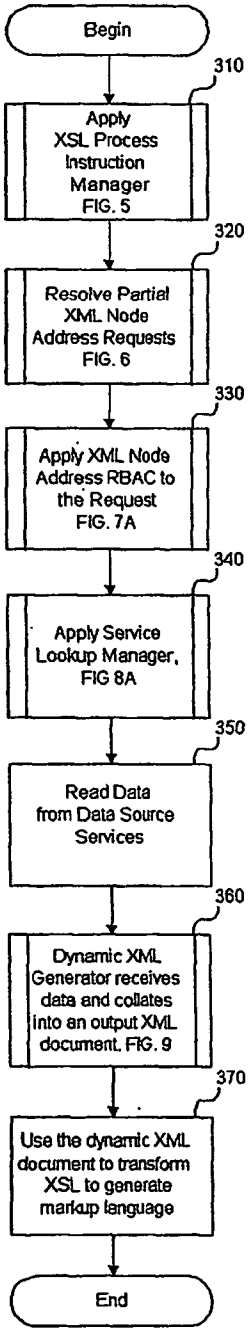


FIG. 3

Write Data Manager

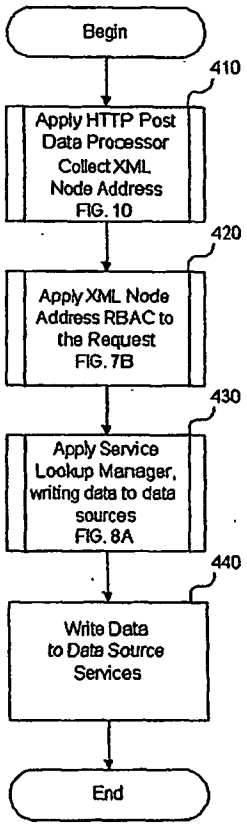


FIG. 4

XSL Process Instruction Manager

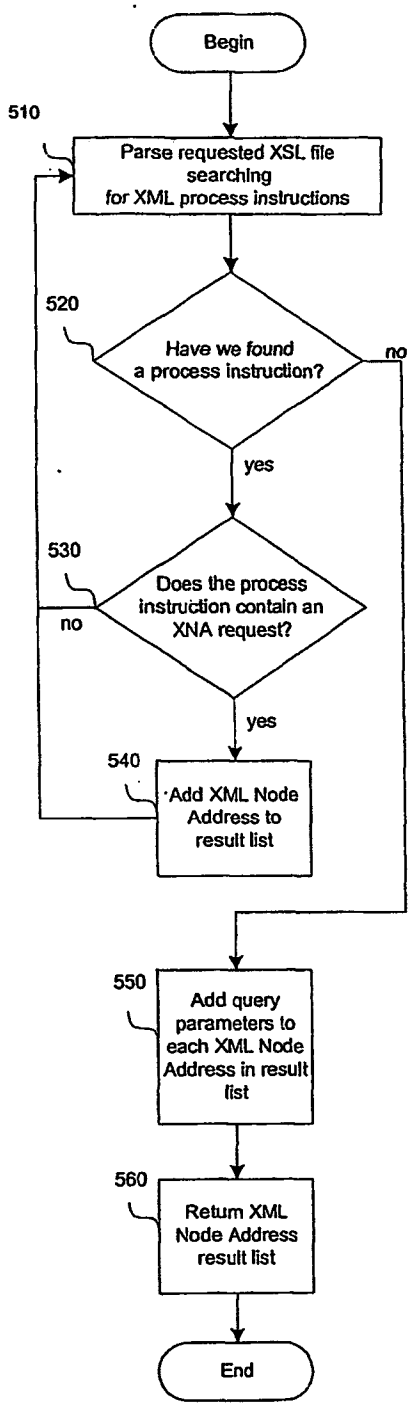


FIG. 5

Service Tree Resolution of Partial XML Node Address

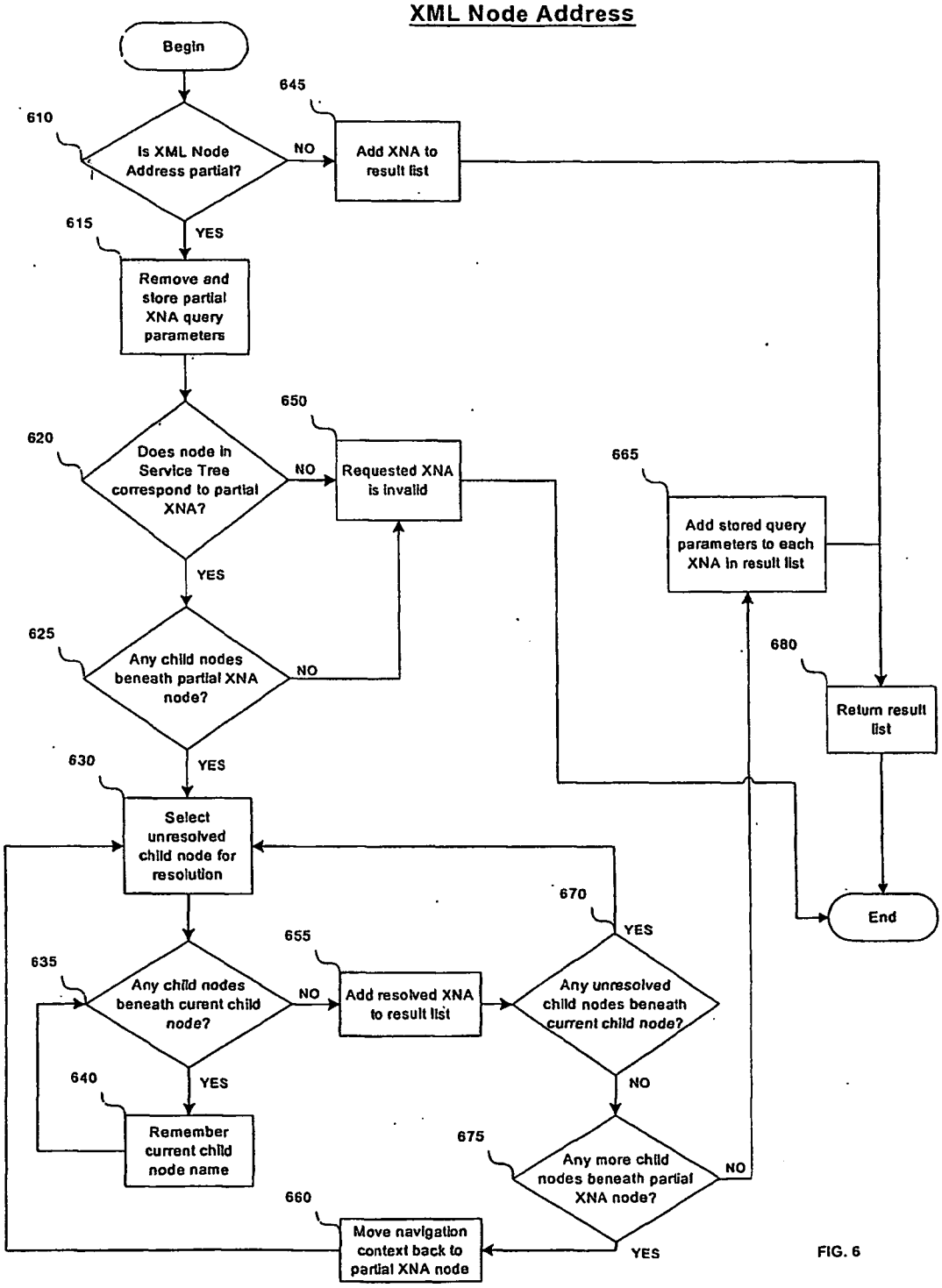


FIG. 6

XML Node Address Role Based Access Control

RBAC while Reading Data

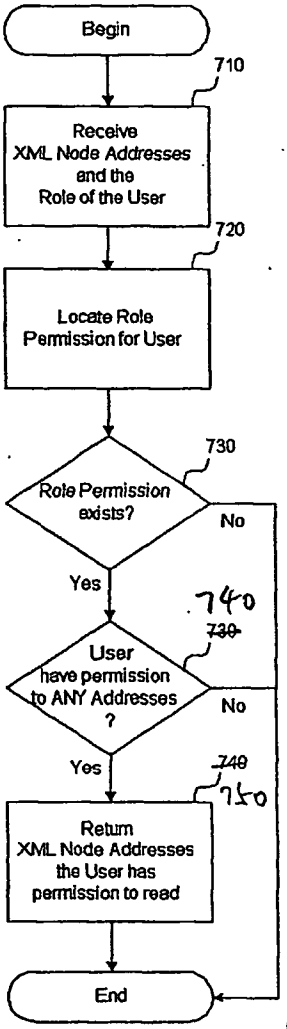


FIG. 7A

RBAC while Writing Data

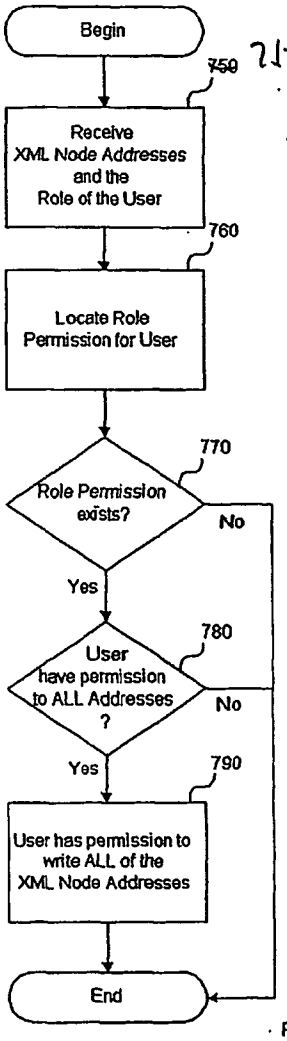


FIG. 7B

Service Lookup Manager

Read/Write Data

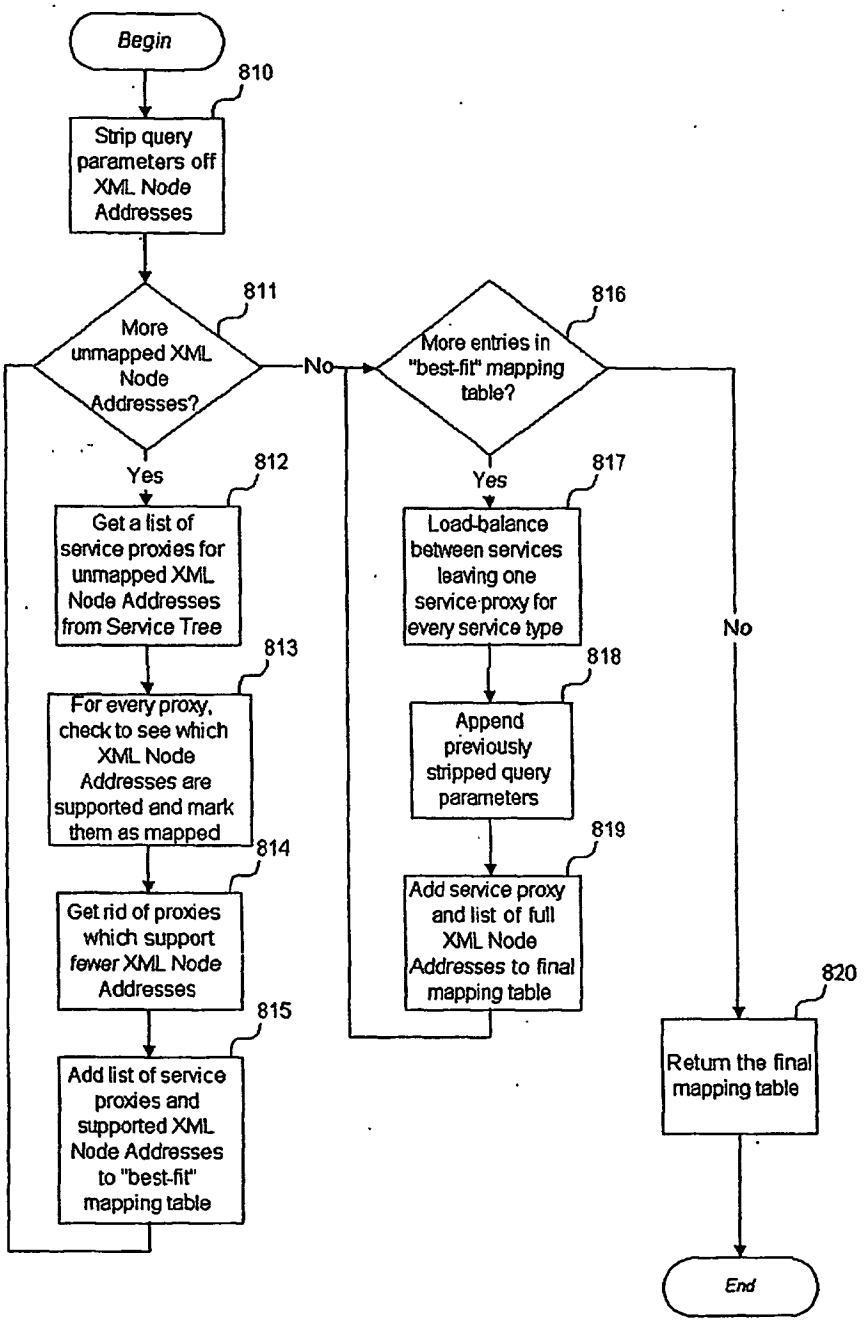


FIG. 8A

Service Lookup Manager

Add Service

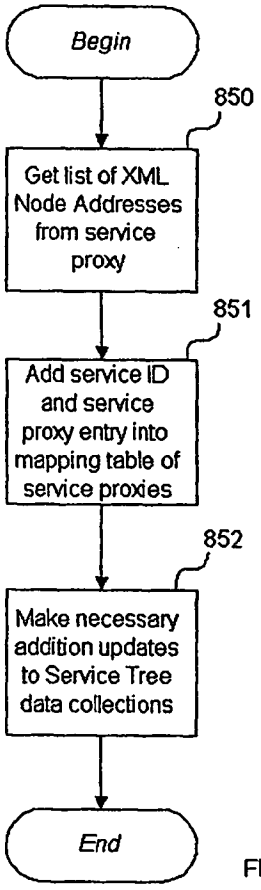


FIG. 8B

Remove Service

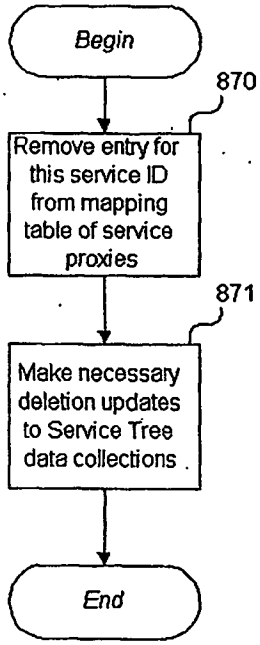


FIG. 8C

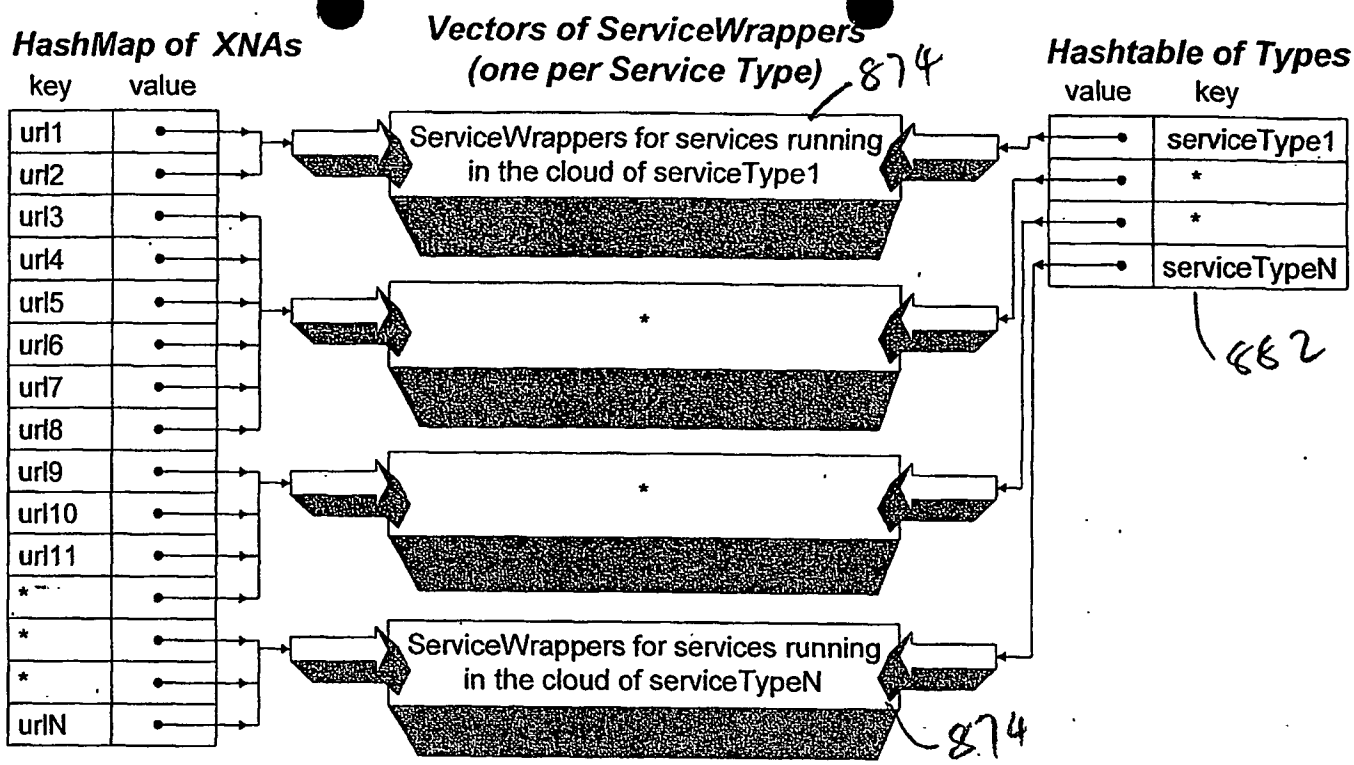
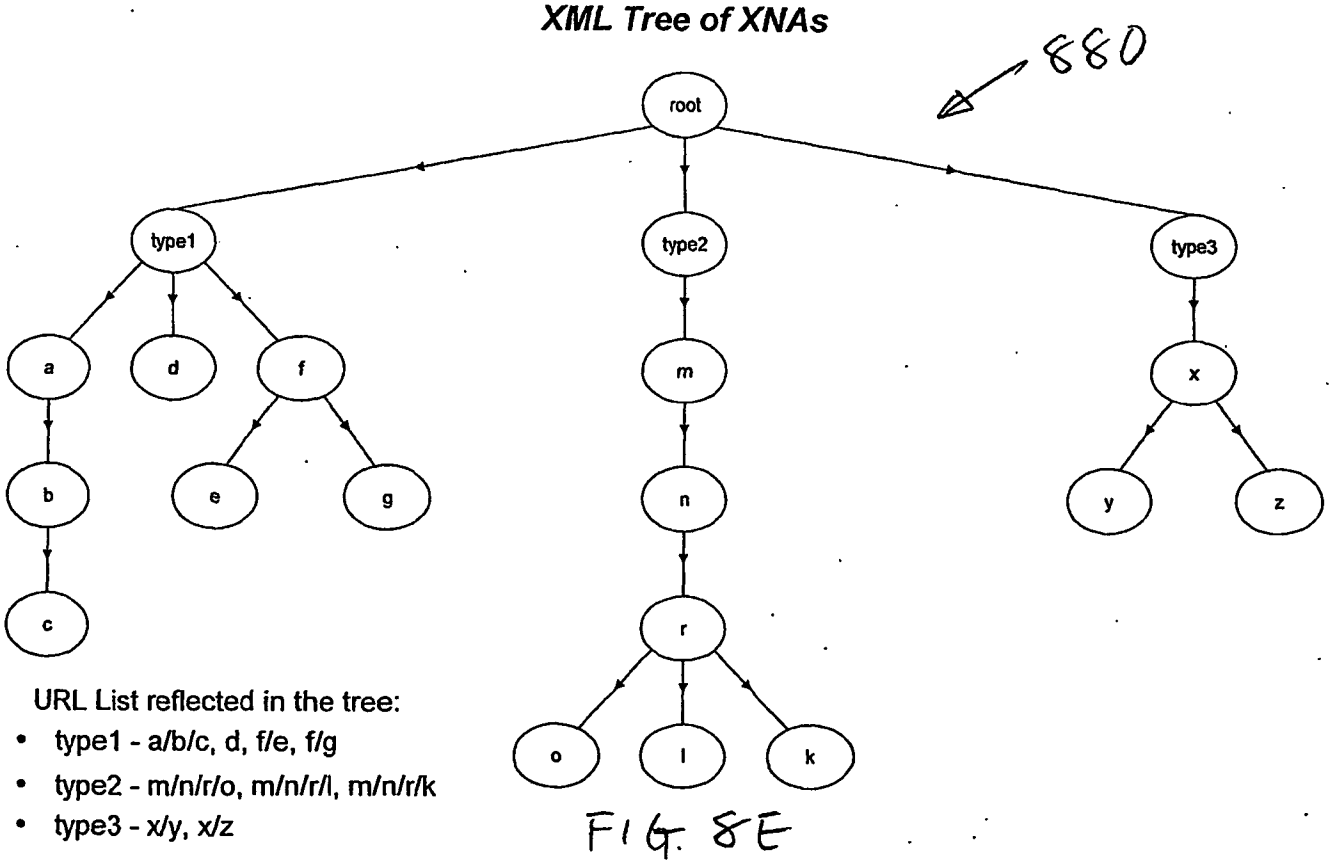


FIG. 8D

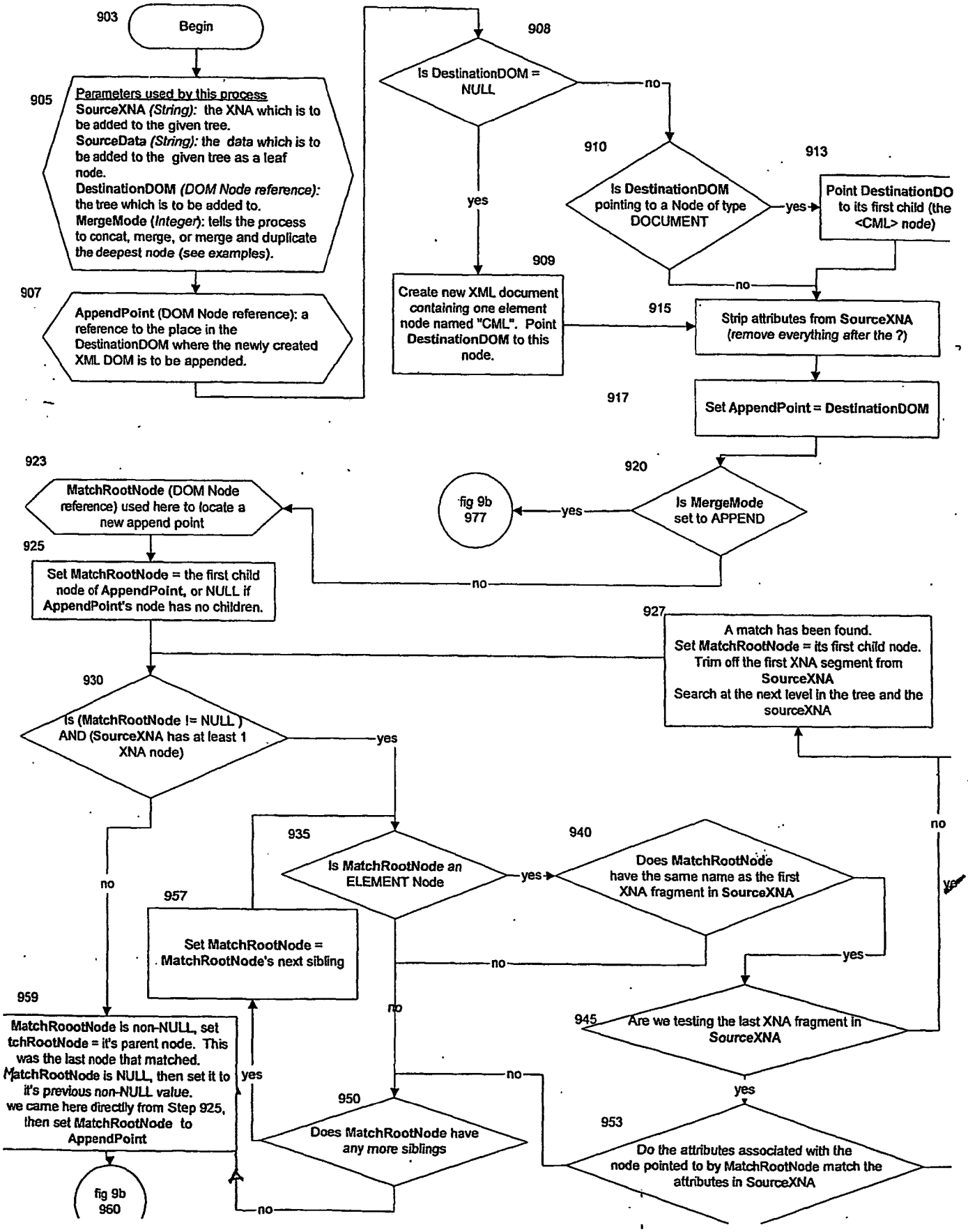


URL List reflected in the tree:

- type1 - a/b/c, d, f/e, f/g
- type2 - m/n/r/o, m/n/r/l, m/n/r/k
- type3 - x/y, x/z

FIG. 8E

Dynamic XML Generator



Dynamic XML Generato

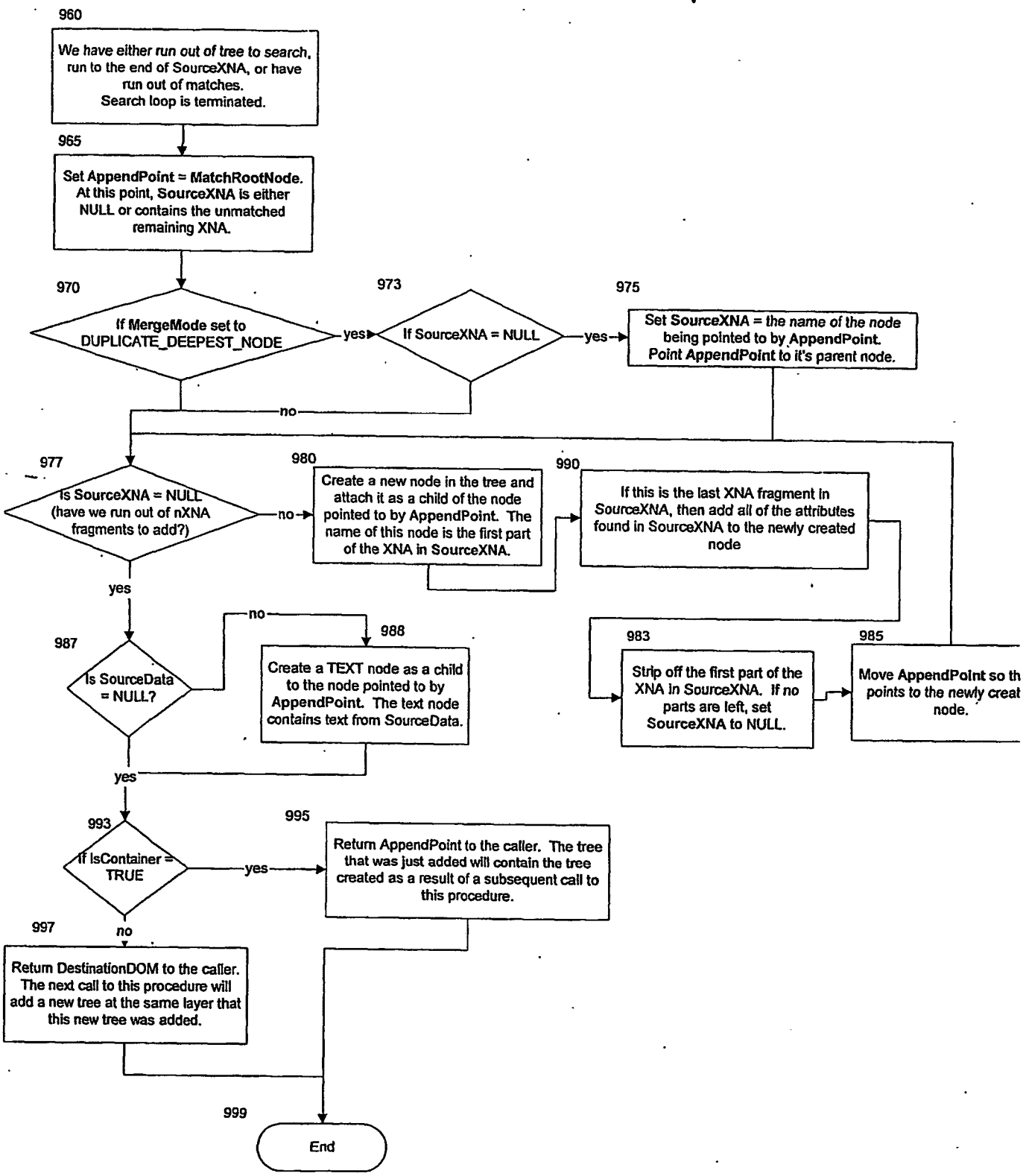


Fig 9B

cml

FIG. 9c

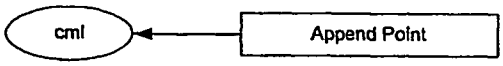


FIG. 9D

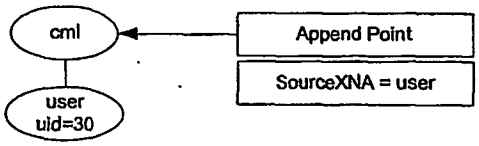


FIG. 9E

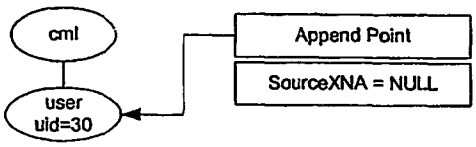


FIG. 9F

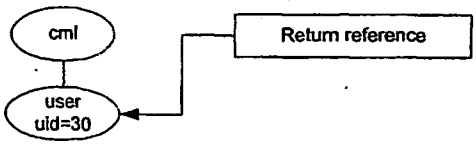


FIG. 9G

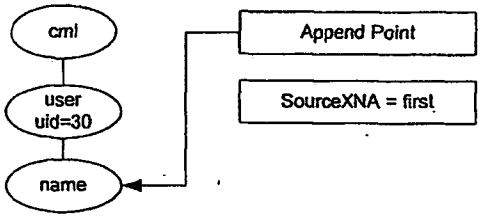


FIG. 9H

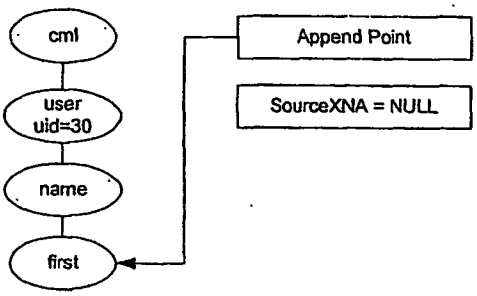


FIG. 9I

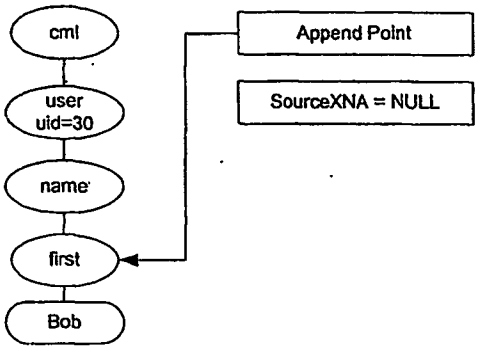


FIG. 9J

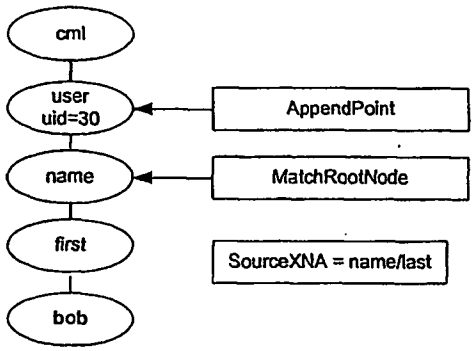

cml
user
uid=30
name
first
bob
AppendPoint
MatchRootNode
SourceXNA = name/last

FIG. 9K

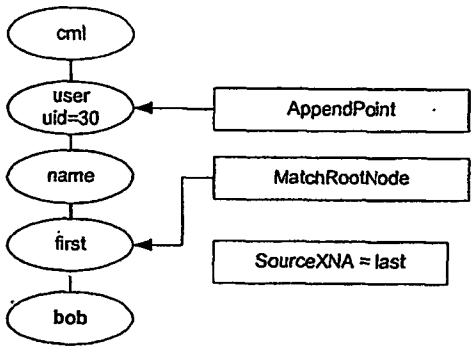

cml
user
uid=30
name
first
bob
AppendPoint
MatchRootNode
SourceXNA = last

FIG. 9L

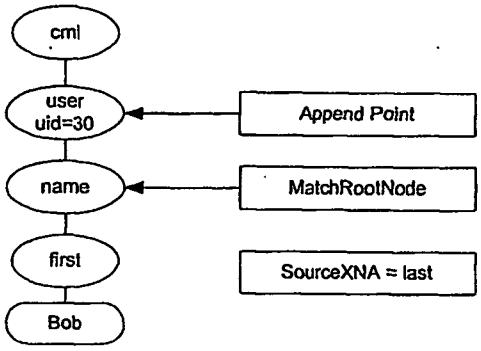

cml
user
uid=30
name
first
Bob
Append Point
MatchRootNode
SourceXNA = last

FIG. 9M

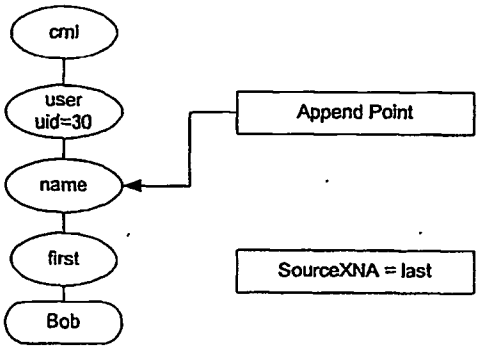


FIG. 9N

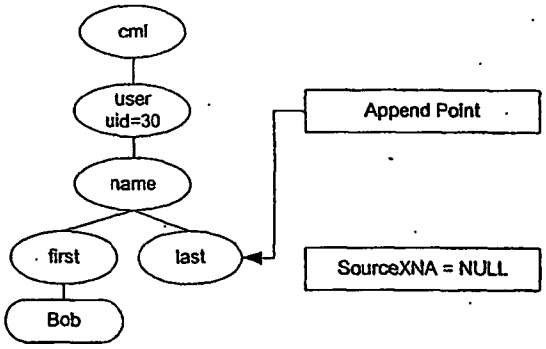

cml
user
uid=30
name
first
last
Bob
Append Point
SourceXNA = NULL

FIG. 90

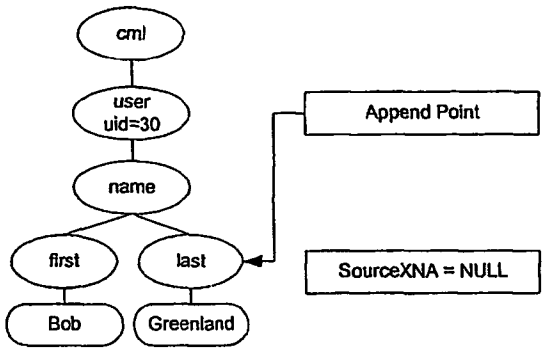

cml
user
uid=30
name
first
last
Bob
Greenland
Append Point
SourceXNA = NULL

FIG. 9P

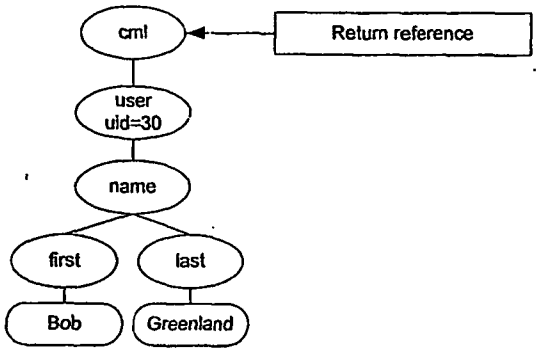

cml
Return reference
user
uid=30
name
first
last
Bob
Greenland

FIG. 9Q

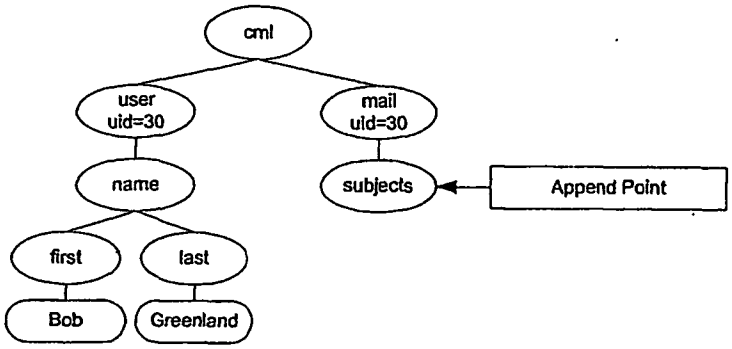

cml
user
uid=30
mail
uid=30
name
subjects
Append Point
first
last
Bob
Greenland

FIG. 9R

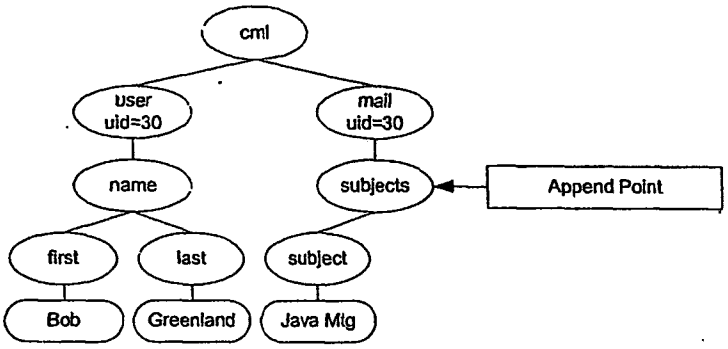

cml
user
uid=30
mail
uid=30
name
subjects
Append Point
first
last
subject
Bob
Greenland
Java Mtg

FIG. 9S

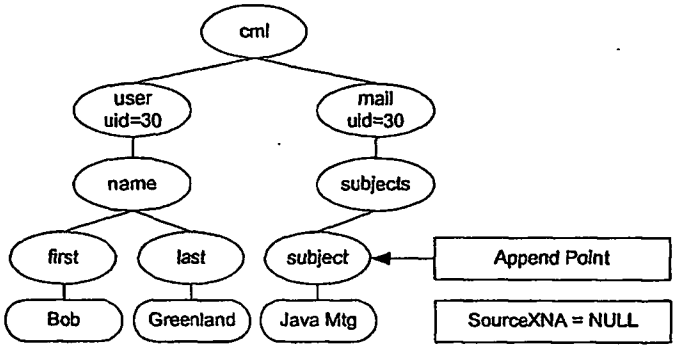

cml
user
uid=30
mail
uid=30
name
subjects
first
last
subject
Append Point
Bob
Greenland
Java Mtg
SourceXNA = NULL

FIG. 9T

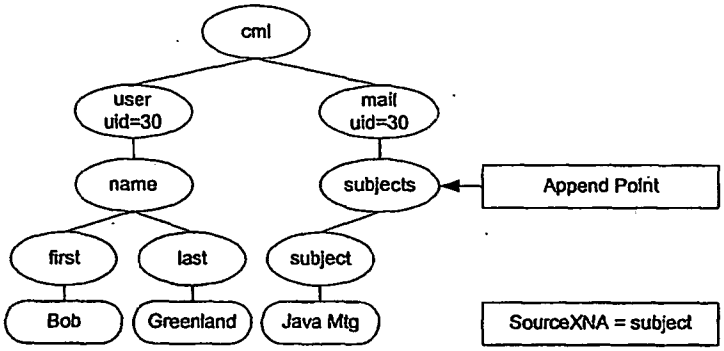

cml
user
uid=30
mail
uid=30
name
subjects
Append Point
first
last
subject
Bob
Greenland
Java Mtg
SourceXNA = subject

FIG. 9U

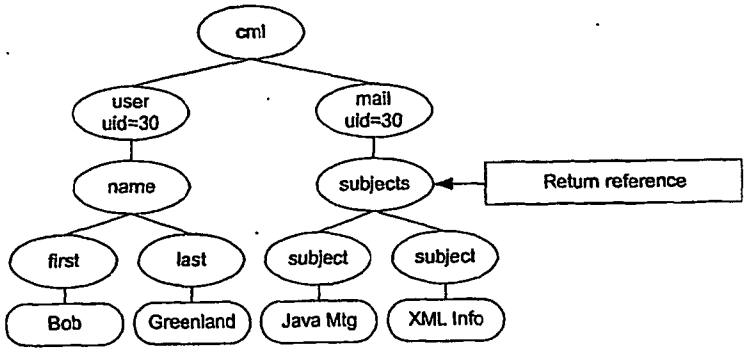

cml
user
uid=30
mail
uid=30
name
subjects
Return reference
first
last
subject
subject
Bob
Greenland
Java Mtg
XML Info

FIG. 9V

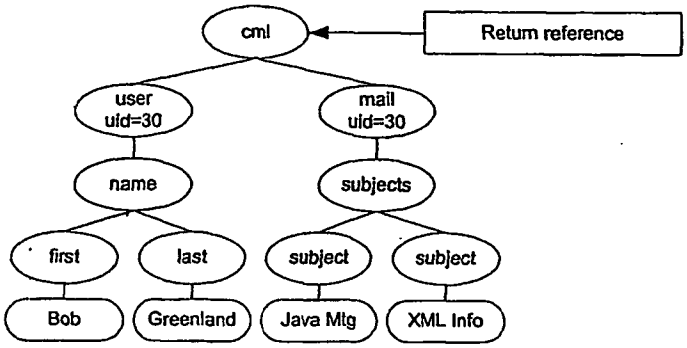

cml
Return reference
user
uid=30
mail
uid=30
name
subjects
first
last
subject
subject
Bob
Greenland
Java Mtg
XML Info

Fig. 9W

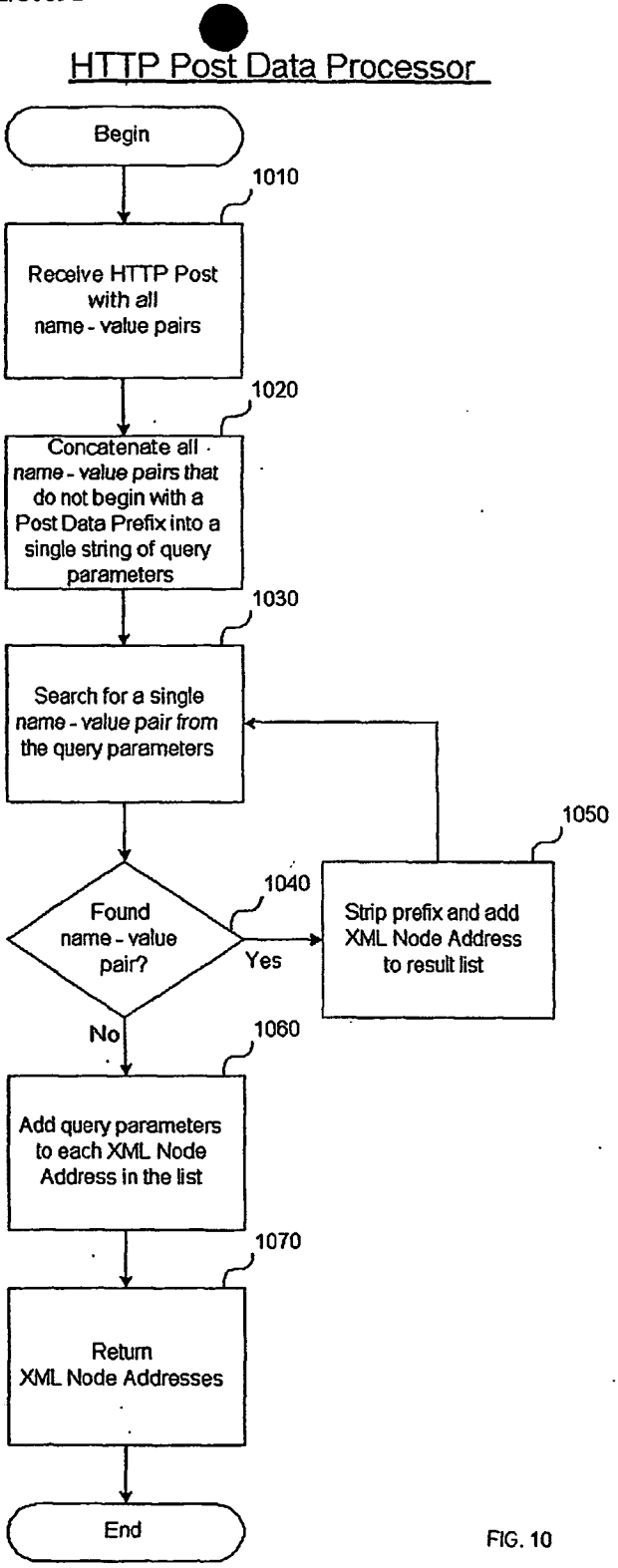

HTTP Post Data Processor
Begin
1010
Receive HTTP Post with all name - value pairs
1020
Concatenate all name - value pairs that do not begin with a Post Data Prefix into a single string of query parameters
1030
Search for a single name - value pair from the query parameters
1050
Found name - value pair?
1040
Yes
Strip prefix and add XML Node Address to result list
No
1060
Add query parameters to each XML Node Address in the list
1070
Return XML Node Addresses
End

FIG. 10

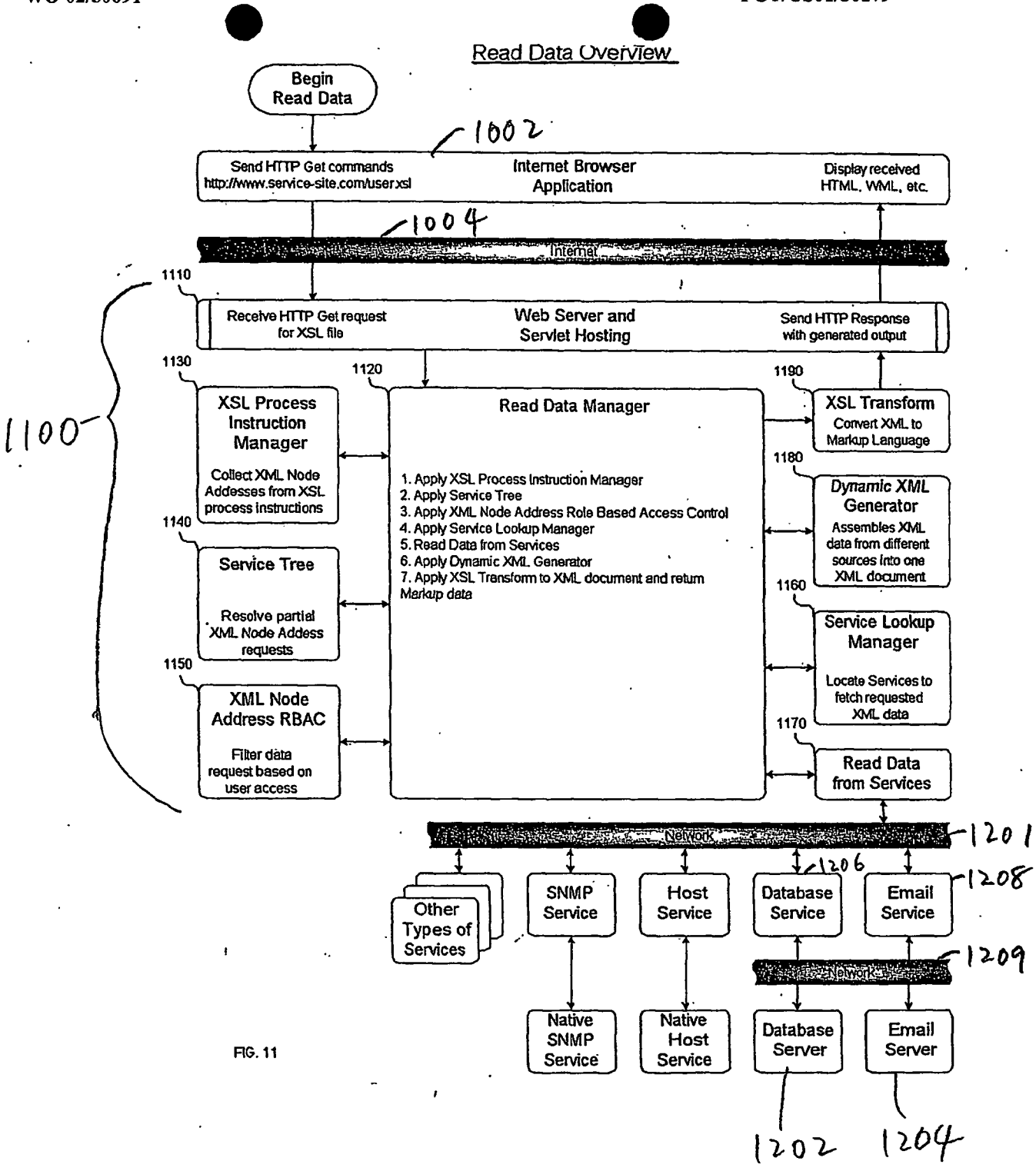

Read Data Overview
Begin Read Data
1002
Send HTTP Get commands http://www.service-site.com/user.xsl
Internet Browser Application
Display received HTML, WML, etc.
1004
Internet
1110
Receive HTTP Get request for XSL file
Web Server and Servlet Hosting
Send HTTP Response with generated output
1100
1130
XSL Process Instruction Manager
Collect XML Node Addresses from XSL process instructions
1120
Read Data Manager
1. Apply XSL Process Instruction Manager
2. Apply Service Tree
3. Apply XML Node Address Role Based Access Control
4. Apply Service Lookup Manager
5. Read Data from Services
6. Apply Dynamic XML Generator
7. Apply XSL Transform to XML document and return Markup data
1190
XSL Transform
Convert XML to Markup Language
1180
Dynamic XML Generator
Assembles XML data from different sources into one XML document
1140
Service Tree
Resolve partial XML Node Addess requests
1160
Service Lookup Manager
Locate Services to fetch requested XML data
1150
XML Node Address RBAC
Filter data request based on user access
1170
Read Data from Services
Network
1201
1206
1208
Other Types of Services
SNMP Service
Host Service
Database Service
Email Service
1209
Network
Native SNMP Service
Native Host Service
Database Server
Email Server
1202
1204

FIG. 11

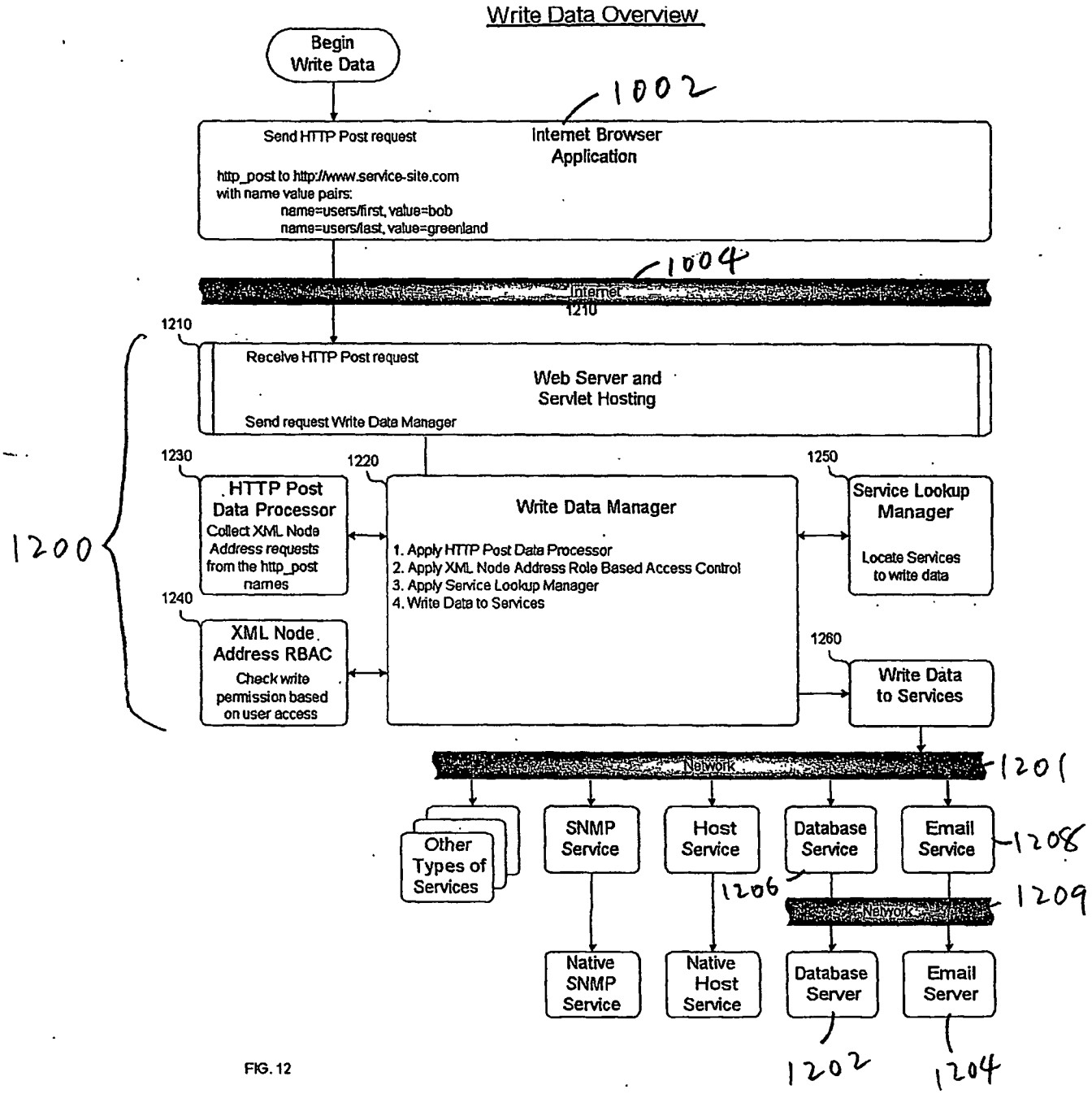
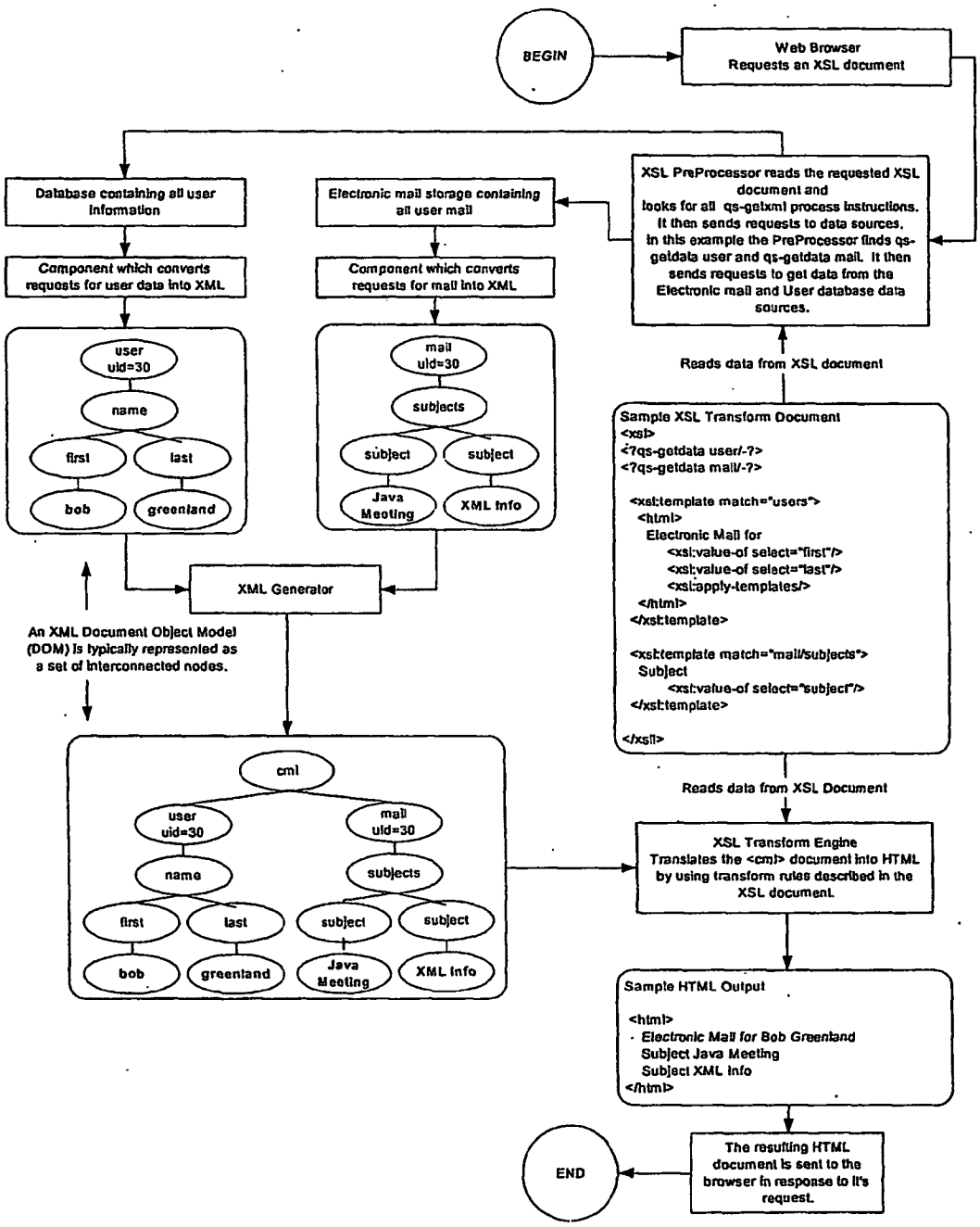

Write Data Overview
Begin Write Data
1002
Send HTTP Post request
Internet Browser Application
http_post to http://www.service-site.com
with name value pairs:
name=users/first, value=bob
name=users/last, value=greenland
1004
Internet
1210
1210
Receive HTTP Post request
Web Server and Servlet Hosting
Send request Write Data Manager
1230
HTTP Post Data Processor
Collect XML Node Address requests from the http_post names
1220
Write Data Manager
1. Apply HTTP Post Data Processor
2. Apply XML Node Address Role Based Access Control
3. Apply Service Lookup Manager
4. Write Data to Services
1250
Service Lookup Manager
Locate Services to write data
1240
XML Node Address RBAC
Check write permission based on user access
1260
Write Data to Services
1200
Network
1201
Other Types of Services
SNMP Service
Host Service
Database Service
Email Service
1208
1206
1209
Network
Native SNMP Service
Native Host Service
Database Server
Email Server
1202
1204

FIG. 12

XML Generator and Process Instruction Manager Example

This example illustrates how the XML Generator is used to prepare XML data from disparate data sources so that it may be merged into a single document and processed by XSL. This examle also shows how the Process Instruction Manager uses XSL process instructions to get XML data from these data sources.



XML Element Role Based Access Control Flow Chart

User role defined as:
XNAPermission "user/-","read";
XNAPermission "user/application","none";

UserAdmin role defined as:
XNAPermission "user/-","read";

This is a flow chart of a read request using the "User" role listed above.

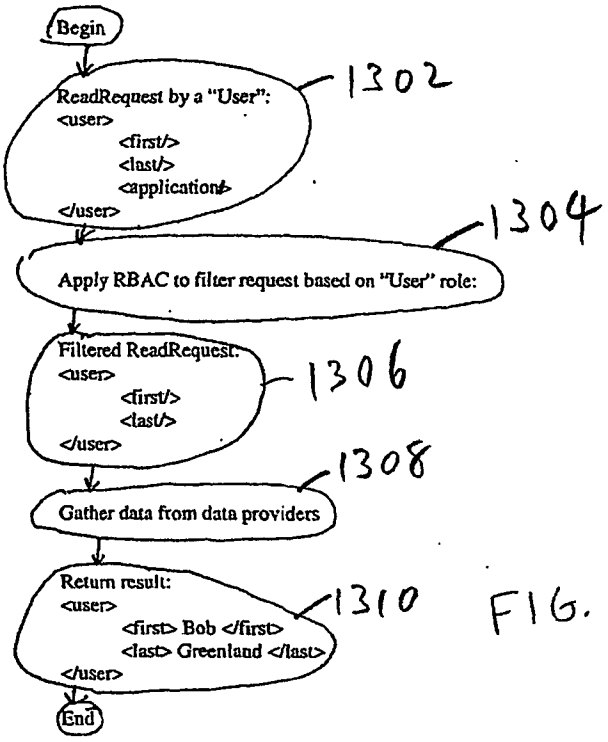


FIG. 14

This is a flow chart of a read request using the "UserAdmin" role listed above. FIG. 2

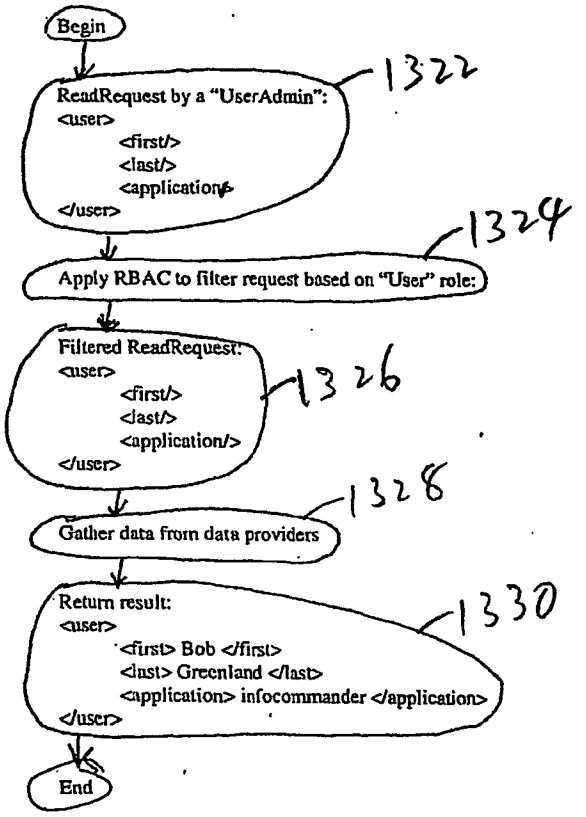


FIG. 15

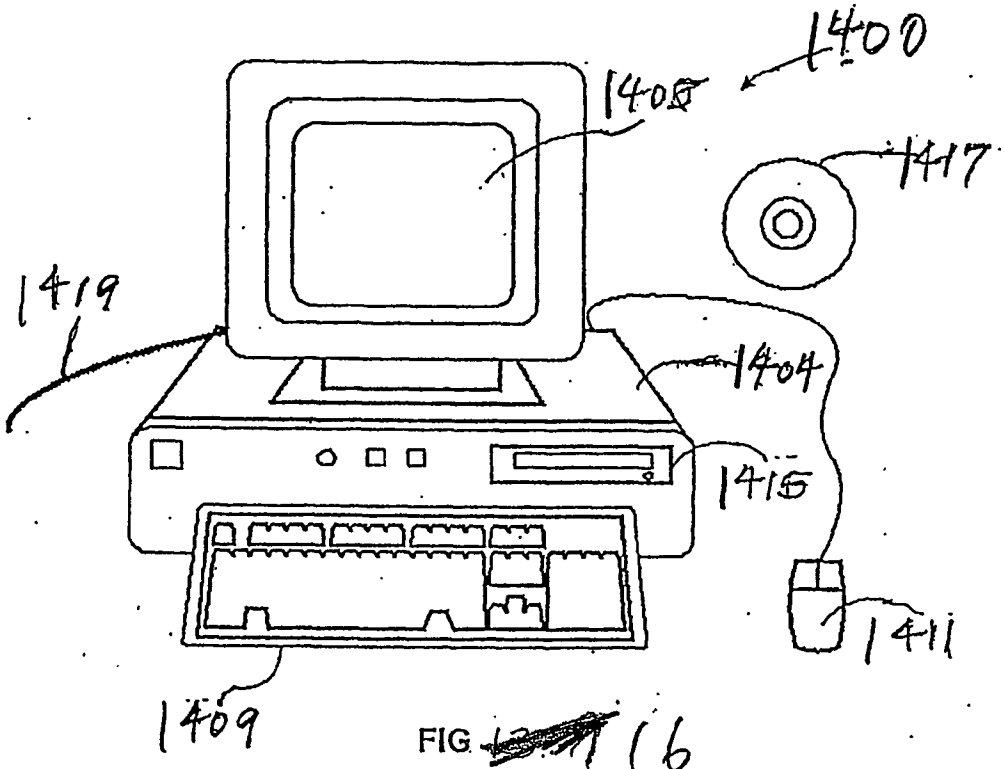


FIG. ~~13~~ 16

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/50279

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G06F 15/00, 17/00, 17/21, 17/24; 17/60
US CL : 707/522, 523, 513; 705/26, 27

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/522, 523, 513; 705/26, 27

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Continuation Sheet

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, P	US 6,209,124 B1 (VERMEIRE et al) 27 March 2001 (27.03.2001), columns 5-7	1-3, 9-10, 12-16, 22, 32-52
A,P	US 6,324,555 B1 (SITES) 27 November 2001 (27.11.2001), column 1, lines 40-60	5-8, 29-34, 40-41
A, E	US 6356920 B1 (VANDERSLUIS) 12 March 2002 (12.03.2002), columns 11-12, 19.	1-4, 9-12, 17-31, 35-38, 42-44, 49-52
A, P	US 6317783 B1 (FREISHTAT et al) 13 November 2001 (13.11.2001), columns 4, 6, and 8	9-10, 35-37, 51-52

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"B" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 April 2002 (11.04.2002)	06 MAY 2002
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Joseph Feild Peggy Harrod Telephone No. 703-305-3900

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

Internal application No.

PCT/US01/50279

Continuation of B. FIELDS SEARCHED Item 3:
WEST
search terms: XML, merge, address, combine, concat, hash, mapping, table

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☑ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☑ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.